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

Semester 202

Assignment

Operating System

Teacher: Nguyen Le Duy Lai
Students: Nguyen Minh Hieu - 1950014
Nguyen Thanh Chuong - 1952595
Nguyen Binh Minh - 1952846

HO CHI MINH CITY, MAY 2021

1 Scheduling

Question

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

Giving each process with a time slot, therefore reducing starvation

In comparision with RR and Priority Queue alone, it is more efficient in terms of average waiting time, turnaround time and numbers of context switches

Implementation

Priority Queue

```
void enqueue(struct queue_t * q, struct pcb_t * proc) {
  /* 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 highestPriorityP = 0, highestPriority = q -> proc[0] -> priority;
    for ( int i = 1; i < q-> size; ++i) {
      if(q -> proc[i]-> priority > highestPriority) {
        highestPriority = q -> proc[i]-> priority;
        highestPriorityP = i;
      }
    }
    struct pcb_t * res = q -> proc[highestPriorityP];

    q -> proc[highestPriorityP] = NULL;

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

  return NULL;
}
```

Get a process from a queue

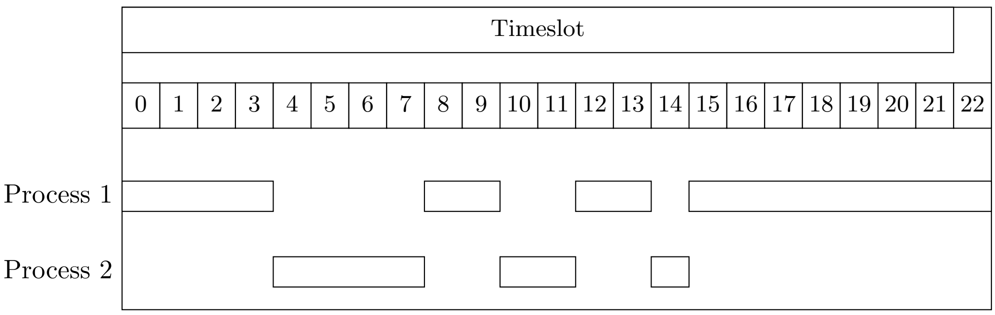
```
struct pcb_t * get_proc(void) {
  struct pcb_t * proc = NULL;
  /*TODO: get a process from [ready_queue]. If ready queue
   * is empty, push all processes in [run_queue] back to
   * [ready_queue] and return the highest priority one.
```

```
     * Remember to use lock to protect the queue.
     * */
    pthread_mutex_lock(&queue_lock);
    if(empty(&ready_queue))
    {
      if(!empty(&run_queue))
      {
        int size = run_queue.size;
        for(int i = 0; i < size; ++i)
        {
          proc = dequeue(&run_queue);
          enqueue(&ready_queue, proc);
        }
        proc = dequeue(&ready_queue);
        //return proc;
      }
    }
    else
    {
      proc = dequeue(&ready_queue);
    }

    pthread_mutex_unlock(&queue_lock);
    return proc;
}
```

Gannt chart

TEST 0



Test 1

As the Gantt-chart is too large, we put it in the end of the report so that it do not ruin the pages

2 Memory

2.1 Question

What is the advantage and disadvantage of segmentation with paging

Advantages

External fragmentation is not there
Segment table has only one entry corresponding to one actual segment

Page table size is limited by segment size ⟶ reduce memory usage
Sharing of data

Disadvantages

It still suffered from internal segmentation
Complexity level is higher than others
Costlier than segmentation and paging

2.2 Result of allocation and deallocation

2.2.1 Test 0

```
- Allocation:
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:
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)
- Deallocation:
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
- Allocation:
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:
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 result:
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)
```

2.2.2 Test 1

```
- Allocation:
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:
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)
- Deallocation:
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
- Allocation:
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:
```

```
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:
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:
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
- Deallocation:

- Final result:
```

2.3 Implementation

2.3.1 Get a page table and translate

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

```
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 address
       * to [p_index] field of page_table->table[i] to
       * produce the correct physical address and save it to
       * [*physical_addr]  */
      *physical_addr = offset + (page_table->table[i].p_index << OFFSET_LEN);
      return 1;
    }
  }
  return 0;
}
```

2.3.2 Allocate Memory

```
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 NumOfPages_avail = 0;
  for(int i = 0; i < NUM_PAGES; ++i)
    if(_mem_stat[i].proc == 0)
      NumOfPages_avail++;

  if(num_pages <= NumOfPages_avail && ((proc->bp - 1024) + num_pages * PAGE_SIZE
    <= RAM_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
     *  - Add entries to segment table page tables of [proc]
```

```
     *    to ensure accesses to allocated memory slot is
     *    valid. */

    //update _ram_stat
    int pageIndex = 0;
    int *arrPhysicalPageIndex = (int*) malloc(sizeof(int) * num_pages);
    addr_t curAddress = ret_mem - 1024;
    for(int i = 0; i < num_pages;++i)
    {
      for(int j = 0; j < NUM_PAGES; ++j)
      {
        if(_mem_stat[j].proc == 0)
        {
          arrPhysicalPageIndex[i] = j;
          _mem_stat[j].proc = proc->pid;
          _mem_stat[j].index = pageIndex; pageIndex ++;
          if(i == num_pages - 1)
            _mem_stat[j].next = -1;
          break;
        }
      }
    }
    for(int i = 0; i < num_pages - 1;++i){
      _mem_stat[arrPhysicalPageIndex[i]].next = arrPhysicalPageIndex[i+1];
    }

    //update multi-level page tables
    proc->seg_table->size = 32;
    for(int i = 0; i < num_pages; ++i){
      addr_t first_level_index = get_first_lv(curAddress);
      addr_t second_level_index = get_second_lv(curAddress);

      if(proc->seg_table->table[first_level_index].pages == NULL){
        struct page_table_t * page_table = (struct page_table_t *)malloc(sizeof(struct page_table_t));
        page_table->size = 32;
        page_table->table[second_level_index].v_index = second_level_index;
        page_table->table[second_level_index].p_index = arrPhysicalPageIndex[i];

        proc->seg_table->table[first_level_index].pages = page_table;
        proc->seg_table->table[first_level_index].v_index = first_level_index;
      }
      else{
        proc->seg_table->table[first_level_index].pages->table[second_level_index].v_index = second_level_index;
        proc->seg_table->table[first_level_index].pages->table[second_level_index].p_index = arrPhysicalPageIndex[i];
      }
      curAddress += 1024;
    }
    free(arrPhysicalPageIndex);
  }

  pthread_mutex_unlock(&mem_lock);
  return ret_mem - 1024;
}
```

2.3.3 Deallocate memory

```
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);
    //update break pointer
    proc->bp = address + 1024;

    //remove pages in RAM
    int num_pages = 1;
    addr_t curAddress = address;
    addr_t physicalAdrress = 0; int physicalIndex = 0;
    translate(curAddress,&physicalAdrress,proc);
    physicalIndex = physicalAdrress / 1024;
    while(_mem_stat[physicalIndex].next != -1)
    {
      _mem_stat[physicalIndex].proc = 0;
      _mem_stat[physicalIndex].index = 0;
      _mem_stat[physicalIndex].next = 0;
      curAddress += 1024;
      translate(curAddress,&physicalAdrress,proc);
      physicalIndex = physicalAdrress / 1024;
      num_pages ++;
    }
    _mem_stat[physicalIndex].proc = 0;
    _mem_stat[physicalIndex].index = 0;
    _mem_stat[physicalIndex].next = 0;

    //remove pages in multi-level page tables
    curAddress = address;
    for(int i = 0; i < num_pages; ++i)
    {
      addr_t first_level_index = get_first_lv(curAddress);
      addr_t second_level_index = get_second_lv(curAddress);
      proc->seg_table->table[first_level_index].pages->table[second_level_index].
      v_index = 0;
      proc->seg_table->table[first_level_index].pages->table[second_level_index].
      p_index = 0;
      curAddress += 1024;
    }

    pthread_mutex_unlock(&mem_lock);
    return 0;
}
```

3 Put it all together

3.1 TEST 0

```
------ OS TEST 0 ---------------------------------------------------------------
./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  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
Time slot  22
  CPU 0: Processed  4 has finished
  CPU 0 stopped

MEMORY CONTENT:
000: 00000-003ff - PID: 03 (idx 000, nxt: 001)
001: 00400-007ff - PID: 03 (idx 001, nxt: 002)
002: 00800-00bff - PID: 03 (idx 002, nxt: 003)
003: 00c00-00fff - PID: 03 (idx 003, nxt: -01)
004: 01000-013ff - PID: 04 (idx 000, nxt: 005)
005: 01400-017ff - PID: 04 (idx 001, nxt: 006)
006: 01800-01bff - PID: 04 (idx 002, 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 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)
023: 05c00-05fff - PID: 02 (idx 000, nxt: 024)
024: 06000-063ff - PID: 02 (idx 001, nxt: 025)
025: 06400-067ff - PID: 02 (idx 002, nxt: 026)
026: 06800-06bff - PID: 02 (idx 003, nxt: -01)
047: 0bc00-0bfff - PID: 01 (idx 000, nxt: -01)
  0bc14: 64
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)
NOTE: Read file output/os_0 to verify your result
```

3.2 TEST 1

```
------ OS TEST 1 ------------------------------------------------------------
./os os_1
Time slot   0
  Loaded a process at input/proc/p0, PID: 1
  CPU 3: Dispatched process  1
Time slot   1
Time slot   2
  Loaded a process at input/proc/s3, PID: 2
Time slot   3
  CPU 0: Dispatched process  2
  CPU 3: Put process  1 to run queue
  CPU 3: Dispatched process  1
Time slot   4
  Loaded a process at input/proc/m1, PID: 3
  CPU 2: Dispatched process  3
Time slot   5
  CPU 0: Put process  2 to run queue
  CPU 0: Dispatched process  2
  CPU 3: Put process  1 to run queue
  CPU 3: Dispatched process  1
Time slot   6
  CPU 2: Put process  3 to run queue
  CPU 2: Dispatched process  3
  Loaded a process at input/proc/s2, PID: 4
Time slot   7
  CPU 0: Put process  2 to run queue
  CPU 0: Dispatched process  4
  CPU 1: Dispatched process  2
  Loaded a process at input/proc/m0, PID: 5
  CPU 3: Put process  1 to run queue
  CPU 3: Dispatched process  5
Time slot   8
  CPU 2: Put process  3 to run queue
  CPU 2: Dispatched process  1
Time slot   9
  Loaded a process at input/proc/p1, PID: 6
  CPU 1: Put process  2 to run queue
  CPU 1: Dispatched process  3
  CPU 0: Put process  4 to run queue
  CPU 0: Dispatched process  6
  CPU 3: Put process  5 to run queue
  CPU 3: Dispatched process  4
Time slot  10
```

```
  CPU 2: Put process  1 to run queue
  CPU 2: Dispatched process  2
Time slot  11
  Loaded a process at input/proc/s0, PID: 7
  CPU 0: Put process  6 to run queue
  CPU 0: Dispatched process  7
  CPU 1: Put process  3 to run queue
  CPU 1: Dispatched process  5
  CPU 3: Put process  4 to run queue
  CPU 3: Dispatched process  4
Time slot  12
  CPU 2: Put process  2 to run queue
  CPU 2: Dispatched process  1
Time slot  13
  CPU 1: Put process  5 to run queue
  CPU 1: Dispatched process  6
  CPU 0: Put process  7 to run queue
  CPU 0: Dispatched process  3
  CPU 3: Put process  4 to run queue
  CPU 3: Dispatched process  4
Time slot  14
  CPU 2: Processed  1 has finished
  CPU 2: Dispatched process  7
Time slot  15
  CPU 3: Put process  4 to run queue
  CPU 3: Dispatched process  2
  CPU 1: Put process  6 to run queue
  CPU 1: Dispatched process  5
  CPU 0: Processed  3 has finished
  CPU 0: Dispatched process  4
Time slot  16
  Loaded a process at input/proc/s1, PID: 8
  CPU 2: Put process  7 to run queue
  CPU 2: Dispatched process  8
Time slot  17
  CPU 0: Put process  4 to run queue
  CPU 0: Dispatched process  6
  CPU 1: Put process  5 to run queue
  CPU 1: Dispatched process  4
  CPU 3: Put process  2 to run queue
  CPU 3: Dispatched process  7
Time slot  18
  CPU 2: Put process  8 to run queue
  CPU 2: Dispatched process  5
Time slot  19
  CPU 3: Put process  7 to run queue
  CPU 2: Processed  5 has finished
  CPU 0: Put process  6 to run queue
  CPU 1: Processed  4 has finished
  CPU 1: Dispatched process  6
  CPU 2: Dispatched process  7
  CPU 3: Dispatched process  8
  CPU 0: Dispatched process  2
Time slot  20
  CPU 0: Processed  2 has finished
  CPU 0 stopped
Time slot  21
  CPU 3: Put process  8 to run queue
  CPU 1: Put process  6 to run queue
  CPU 1: Dispatched process  6
  CPU 2: Put process  7 to run queue
  CPU 2: Dispatched process  7
```

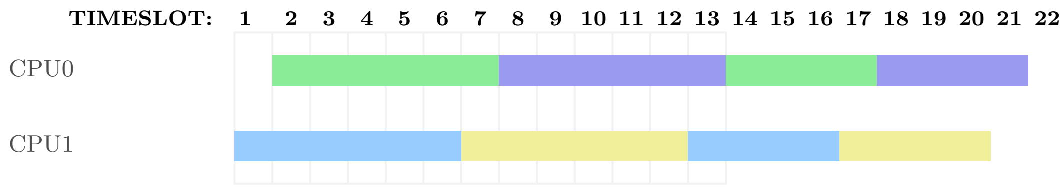
```
  CPU 3: Dispatched process  8
Time slot  22
Time slot  23
  CPU 3: Put process  8 to run queue
  CPU 3: Dispatched process  8
  CPU 2: Put process  7 to run queue
  CPU 2: Dispatched process  7
  CPU 1: Processed  6 has finished
  CPU 1 stopped
Time slot  24
  CPU 3: Processed  8 has finished
  CPU 3 stopped
Time slot  25
  CPU 2: Put process  7 to run queue
  CPU 2: Dispatched process  7
Time slot  26
Time slot  27
  CPU 2: Put process  7 to run queue
  CPU 2: Dispatched process  7
Time slot  28
  CPU 2: Processed  7 has finished
  CPU 2 stopped

MEMORY CONTENT:
000: 00000-003ff - PID: 05 (idx 000, nxt: 001)
  003e8: 15
001: 00400-007ff - PID: 05 (idx 001, nxt: -01)
002: 00800-00bff - PID: 05 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 05 (idx 001, nxt: 004)
004: 01000-013ff - PID: 05 (idx 002, nxt: 005)
005: 01400-017ff - PID: 05 (idx 003, nxt: 006)
006: 01800-01bff - PID: 05 (idx 004, nxt: -01)
011: 02c00-02fff - PID: 06 (idx 000, nxt: 012)
012: 03000-033ff - PID: 06 (idx 001, nxt: 013)
013: 03400-037ff - PID: 06 (idx 002, nxt: 014)
014: 03800-03bff - PID: 06 (idx 003, nxt: -01)
021: 05400-057ff - PID: 01 (idx 000, nxt: -01)
  05414: 64
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 000, nxt: 032)
032: 08000-083ff - PID: 06 (idx 001, nxt: 033)
033: 08400-087ff - PID: 06 (idx 002, nxt: 034)
  085e7: 0a
034: 08800-08bff - PID: 06 (idx 003, nxt: 035)
035: 08c00-08fff - PID: 06 (idx 004, nxt: -01)
NOTE: Read file output/os_1 to verify your result
```

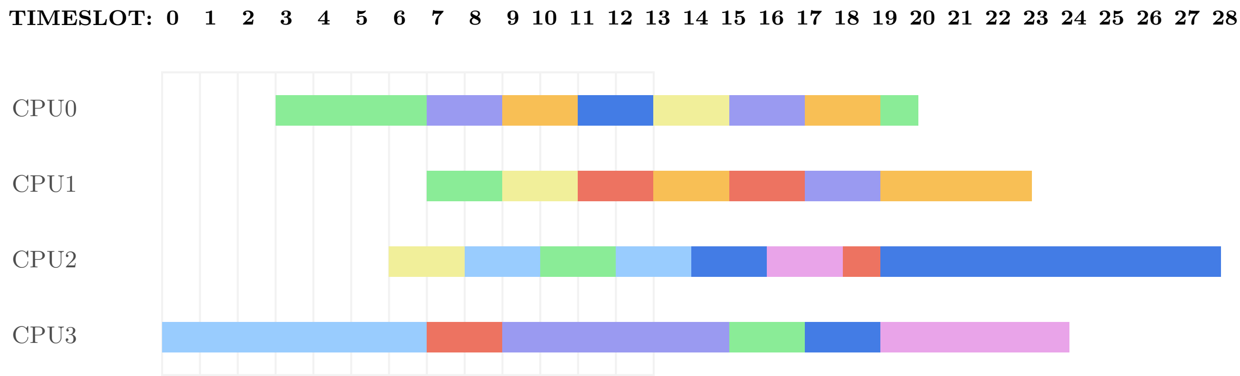
Because the loader and the scheduler run concurrently, our result might be different from the result from output folder

Gantt-chart

Test 0

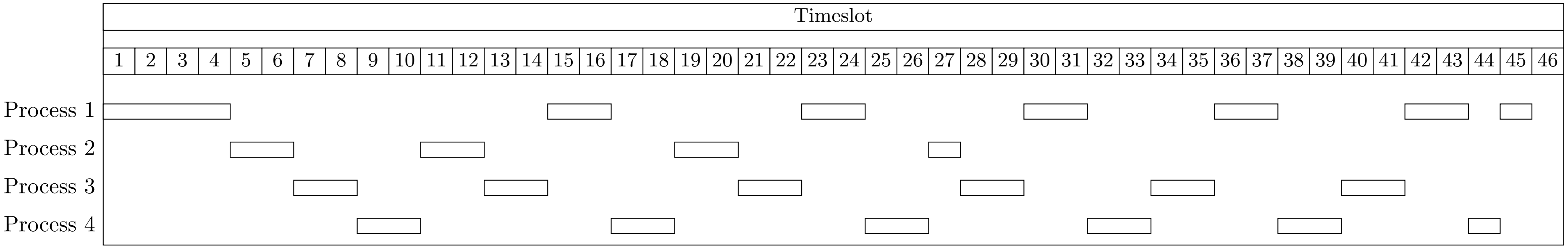


Test 1



Note

Process 1 Process 2 Process 3 Process 4 Process 5 Process 6 Process 7 Process 8


Timeslot
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 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
Process 1
Process 2
Process 3
Process 4