

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 comparison 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

```
1 void enqueue(struct queue_t * q, struct pcb_t * proc) {
2     /* put a new process to queue [q] */
3     if(q->size < MAX_QUEUE_SIZE) {
4         q->proc[q->size] = proc;
5         q->size ++;
6     }
7 }

1 struct pcb_t * dequeue(struct queue_t * q) {
2     /* TODO: return a pcb whose priority is the highest
3      * in the queue [q] and remember to remove it from q
4      */
5     if(!empty(q))
6     {
7         int highestPriorityP = 0, highestPriority = q->proc[0]->priority;
8         for (int i = 1; i < q->size; ++i) {
9             if(q->proc[i]->priority > highestPriority) {
10                 highestPriority = q->proc[i]->priority;
11                 highestPriorityP = i;
12             }
13         }
14         struct pcb_t * res = q->proc[highestPriorityP];
15
16         q->proc[highestPriorityP] = NULL;
17
18         for (int i = highestPriorityP; i < q->size - 1; ++i) {
19             q->proc[i] = q->proc[i+1];
20         }
21         q->size --;
22         return res;
23     }
24
25     return NULL;
26 }
```

Get a process from a queue

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

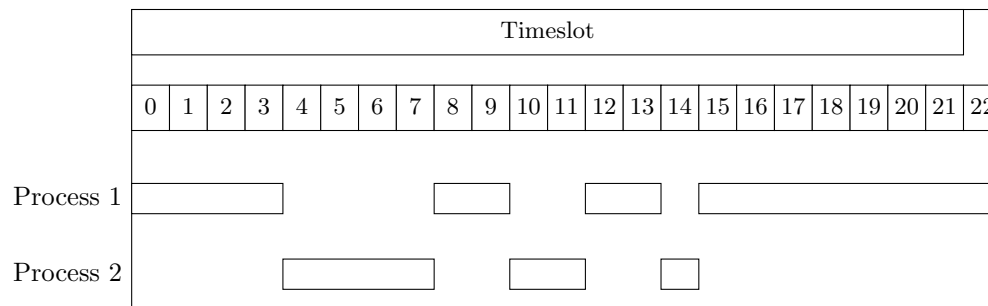
```

6  * Remember to use lock to protect the queue.
7  * */
8  pthread_mutex_lock(&queue_lock);
9  if (empty(&ready_queue))
10 {
11     if (!empty(&run_queue))
12     {
13         int size = run_queue.size;
14         for (int i = 0; i < size; ++i)
15         {
16             proc = dequeue(&run_queue);
17             enqueue(&ready_queue, proc);
18         }
19         proc = dequeue(&ready_queue);
20         //return proc;
21     }
22 }
23 else
24 {
25     proc = dequeue(&ready_queue);
26 }
27
28 pthread_mutex_unlock(&queue_lock);
29 return proc;
30 }

```

Gantt 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

```
1 - Allocation:
2 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
3 001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
4 002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
5 003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
6 004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
7 005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
8 006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
9 007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
10 008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
11 009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
12 010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
13 011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
14 012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
15 013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
16 - Allocation:
17 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
18 001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
19 002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
20 003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
21 004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
22 005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
23 006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
24 007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
25 008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
26 009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
27 010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
28 011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
29 012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
30 013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
31 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
32 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
33 - Deallocation:
34 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
35 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
36 - Allocation:
37 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
38 001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
39 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
40 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
41 - Allocation:
42 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
43 001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
44 002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
45 003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
46 004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
```



```
47 005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
48 006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
49 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
50 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
51 - Final result:
52 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
53 003e8: 15
54 001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
55 002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
56 003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
57 004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
58 005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
59 006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
60 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
61 03814: 66
62 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
```

2.2.2 Test 1

```
1 - Allocation:
2 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
3 001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
4 002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
5 003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
6 004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
7 005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
8 006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
9 007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
10 008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
11 009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
12 010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
13 011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
14 012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
15 013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
16 - Allocation:
17 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
18 001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
19 002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
20 003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
21 004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
22 005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
23 006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
24 007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
25 008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
26 009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
27 010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
28 011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
29 012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
30 013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
31 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
32 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
33 - Deallocation:
34 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
35 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
36 - Allocation:
37 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
38 001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
39 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
40 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
41 - Allocation:
```

```
42 000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
43 001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
44 002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
45 003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
46 004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
47 005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
48 006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
49 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
50 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
51 - Deallocation:
52 002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
53 003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
54 004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
55 005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
56 006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
57 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
58 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
59 - Deallocation:
60 014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
61 015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
62 - Deallocation:
63
64 - Final result:
```

2.3 Implementation

2.3.1 Get a page table and translate

```
1 static struct page_table_t * get_page_table(
2     addr_t index, // Segment level index
3     struct seg_table_t * seg_table) { // first level table
4
5     /*
6      * TODO: Given the Segment index [index], you must go through each
7      * row of the segment table [seg_table] and check if the v_index
8      * field of the row is equal to the index
9      *
10     */
11
12     int i;
13     for (i = 0; i < seg_table->size; i++) {
14         // Enter your code here
15         if (seg_table->table[i].v_index == index)
16             return seg_table->table[i].pages;
17     }
18     return NULL;
19 }
20
21 static int translate(
22     addr_t virtual_addr, // Given virtual address
23     addr_t * physical_addr, // Physical address to be returned
24     struct pcb_t * proc) { // Process uses given virtual address
25
26     /* Offset of the virtual address */
27     addr_t offset = get_offset(virtual_addr);
28     /* The first layer index */
29     addr_t first_lv = get_first_lv(virtual_addr);
30     /* The second layer index */
31     addr_t second_lv = get_second_lv(virtual_addr);
32 }
```

```
13  /* Search in the first level */
14  struct page_table_t * page_table = NULL;
15  page_table = get_page_table(first_lv, proc->seg_table);
16  if (page_table == NULL) {
17      return 0;
18  }
19
20  int i;
21  for (i = 0; i < page_table->size; i++) {
22      if (page_table->table[i].v_index == second_lv) {
23          /* TODO: Concatenate the offset of the virtual address
24           * to [p_index] field of page_table->table[i] to
25           * produce the correct physical address and save it to
26           * [*physical_addr] */
27          *physical_addr = offset + (page_table->table[i].p_index << OFFSET_LEN);
28          return 1;
29      }
30  }
31  return 0;
32 }
```

2.3.2 Allocate Memory

```
1  addr_t alloc_mem(uint32_t size, struct pcb_t * proc) {
2      pthread_mutex_lock(&mem_lock);
3      addr_t ret_mem = 0;
4      /* TODO: Allocate [size] byte in the memory for the
5       * process [proc] and save the address of the first
6       * byte in the allocated memory region to [ret_mem].
7       * */
8
9      uint32_t num_pages = (size % PAGE_SIZE) ? size / PAGE_SIZE + 1:
10         size / PAGE_SIZE; // Number of pages we will use
11      int mem_avail = 0; // We could allocate new memory region or not?
12
13      /* First we must check if the amount of free memory in
14       * virtual address space and physical address space is
15       * large enough to represent the amount of required
16       * memory. If so, set 1 to [mem_avail].
17       * Hint: check [proc] bit in each page of _mem_stat
18       * to know whether this page has been used by a process.
19       * For virtual memory space, check bp (break pointer).
20       * */
21      int NumOfPages_avail = 0;
22      for (int i = 0; i < NUM_PAGES; ++i)
23          if (_mem_stat[i].proc == 0)
24              NumOfPages_avail++;
25
26      if (num_pages <= NumOfPages_avail && ((proc->bp - 1024) + num_pages * PAGE_SIZE
27         <= RAM_SIZE))
28          mem_avail = 1;
29
30      if (mem_avail) {
31          /* We could allocate new memory region to the process */
32          ret_mem = proc->bp;
33          proc->bp += num_pages * PAGE_SIZE;
34          /* Update status of physical pages which will be allocated
35           * to [proc] in _mem_stat. Tasks to do:
36           * - Update [proc], [index], and [next] field
37           * - Add entries to segment table page tables of [proc]
```

```
37      *   to ensure accesses to allocated memory slot is
38      *   valid. */
39
40      //update _ram_stat
41      int pageIndex = 0;
42      int *arrPhysicalPageIndex = (int*) malloc(sizeof(int) * num_pages);
43      addr_t curAddress = ret_mem - 1024;
44      for(int i = 0; i < num_pages; ++i)
45      {
46          for(int j = 0; j < NUM_PAGES; ++j)
47          {
48              if(_mem_stat[j].proc == 0)
49              {
50                  arrPhysicalPageIndex[i] = j;
51                  _mem_stat[j].proc = proc->pid;
52                  _mem_stat[j].index = pageIndex; pageIndex ++;
53                  if(i == num_pages - 1)
54                      _mem_stat[j].next = -1;
55                  break;
56              }
57          }
58      }
59      for(int i = 0; i < num_pages - 1; ++i){
60          _mem_stat[arrPhysicalPageIndex[i]].next = arrPhysicalPageIndex[i + 1];
61      }
62
63      //update multi-level page tables
64      proc->seg_table->size = 32;
65      for(int i = 0; i < num_pages; ++i){
66          addr_t first_level_index = get_first_lv(curAddress);
67          addr_t second_level_index = get_second_lv(curAddress);
68
69          if(proc->seg_table->table[first_level_index].pages == NULL){
70              struct page_table_t * page_table = (struct page_table_t *) malloc(sizeof(
71              struct page_table_t));
72              page_table->size = 32;
73              page_table->table[second_level_index].v_index = second_level_index;
74              page_table->table[second_level_index].p_index = arrPhysicalPageIndex[i];
75
76              proc->seg_table->table[first_level_index].pages = page_table;
77              proc->seg_table->table[first_level_index].v_index = first_level_index;
78          }
79          else{
80              proc->seg_table->table[first_level_index].pages->table[second_level_index]
81              .v_index = second_level_index;
82              proc->seg_table->table[first_level_index].pages->table[second_level_index]
83              .p_index = arrPhysicalPageIndex[i];
84          }
85          curAddress += 1024;
86      }
87      free(arrPhysicalPageIndex);
88  }
89
90  pthread_mutex_unlock(&mem_lock);
91  return ret_mem - 1024;
92 }
```

2.3.3 Deallocate memory

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



```

2  /*TODO: Release memory region allocated by [proc]. The first byte of
3  * this region is indicated by [address]. Task to do:
4  * - Set flag [proc] of physical page use by the memory block
5  *   back to zero to indicate that it is free.
6  * - Remove unused entries in segment table and page tables of
7  *   the process [proc].
8  * - Remember to use lock to protect the memory from other
9  *   processes. */
10 pthread_mutex_lock(&mem_lock);
11 //update break pointer
12 proc->bp = address + 1024;
13
14 //remove pages in RAM
15 int num_pages = 1;
16 addr_t curAddress = address;
17 addr_t physicalAddress = 0; int physicalIndex = 0;
18 translate(curAddress,&physicalAddress,proc);
19 physicalIndex = physicalAddress / 1024;
20 while(_mem_stat[physicalIndex].next != -1)
21 {
22     _mem_stat[physicalIndex].proc = 0;
23     _mem_stat[physicalIndex].index = 0;
24     _mem_stat[physicalIndex].next = 0;
25     curAddress += 1024;
26     translate(curAddress,&physicalAddress,proc);
27     physicalIndex = physicalAddress / 1024;
28     num_pages++;
29 }
30 _mem_stat[physicalIndex].proc = 0;
31 _mem_stat[physicalIndex].index = 0;
32 _mem_stat[physicalIndex].next = 0;
33
34 //remove pages in multi-level page tables
35 curAddress = address;
36 for(int i = 0; i < num_pages; ++i)
37 {
38     addr_t first_level_index = get_first_lv(curAddress);
39     addr_t second_level_index = get_second_lv(curAddress);
40     proc->seg_table->table[first_level_index].pages->table[second_level_index].
41     v_index = 0;
42     proc->seg_table->table[first_level_index].pages->table[second_level_index].
43     p_index = 0;
44     curAddress += 1024;
45 }
46 pthread_mutex_unlock(&mem_lock);
47 return 0;
48 }

```

3 Put it all together

3.1 TEST 0

```

1  ----- OS TEST 0 -----
2  ./os os_0
3  Time slot    0
4  Loaded a process at input/proc/p0, PID: 1
5  Time slot    1
6  CPU 1: Dispatched process 1

```



```
7 Time slot 2
8   Loaded a process at input/proc/p1, PID: 2
9   CPU 0: Dispatched process 2
10 Time slot 3
11   Loaded a process at input/proc/p1, PID: 3
12 Time slot 4
13   Loaded a process at input/proc/p1, PID: 4
14 Time slot 5
15 Time slot 6
16 Time slot 7
17   CPU 1: Put process 1 to run queue
18   CPU 1: Dispatched process 3
19 Time slot 8
20   CPU 0: Put process 2 to run queue
21   CPU 0: Dispatched process 4
22 Time slot 9
23 Time slot 10
24 Time slot 11
25 Time slot 12
26 Time slot 13
27   CPU 1: Put process 3 to run queue
28   CPU 1: Dispatched process 1
29 Time slot 14
30   CPU 0: Put process 4 to run queue
31   CPU 0: Dispatched process 2
32 Time slot 15
33 Time slot 16
34 Time slot 17
35   CPU 1: Processed 1 has finished
36   CPU 1: Dispatched process 3
37 Time slot 18
38   CPU 0: Processed 2 has finished
39   CPU 0: Dispatched process 4
40 Time slot 19
41 Time slot 20
42 Time slot 21
43   CPU 1: Processed 3 has finished
44   CPU 1 stopped
45 Time slot 22
46   CPU 0: Processed 4 has finished
47   CPU 0 stopped
48
49 MEMORY CONTENT:
50 000: 00000-003ff - PID: 03 (idx 000, nxt: 001)
51 001: 00400-007ff - PID: 03 (idx 001, nxt: 002)
52 002: 00800-00bff - PID: 03 (idx 002, nxt: 003)
53 003: 00c00-00fff - PID: 03 (idx 003, nxt: -01)
54 004: 01000-013ff - PID: 04 (idx 000, nxt: 005)
55 005: 01400-017ff - PID: 04 (idx 001, nxt: 006)
56 006: 01800-01bff - PID: 04 (idx 002, nxt: 012)
57 007: 01c00-01fff - PID: 02 (idx 000, nxt: 008)
58 008: 02000-023ff - PID: 02 (idx 001, nxt: 009)
59 009: 02400-027ff - PID: 02 (idx 002, nxt: 010)
60   025e7: 0a
61 010: 02800-02bff - PID: 02 (idx 003, nxt: 011)
62 011: 02c00-02fff - PID: 02 (idx 004, nxt: -01)
63 012: 03000-033ff - PID: 04 (idx 003, nxt: -01)
64 014: 03800-03bff - PID: 03 (idx 000, nxt: 015)
65 015: 03c00-03fff - PID: 03 (idx 001, nxt: 016)
66 016: 04000-043ff - PID: 03 (idx 002, nxt: 017)
67   041e7: 0a
68 017: 04400-047ff - PID: 03 (idx 003, nxt: 018)
```



```
69 018: 04800-04bff - PID: 03 (idx 004, nxt: -01)
70 023: 05c00-05fff - PID: 02 (idx 000, nxt: 024)
71 024: 06000-063ff - PID: 02 (idx 001, nxt: 025)
72 025: 06400-067ff - PID: 02 (idx 002, nxt: 026)
73 026: 06800-06bff - PID: 02 (idx 003, nxt: -01)
74 047: 0bc00-0bfff - PID: 01 (idx 000, nxt: -01)
75 0bc14: 64
76 057: 0e400-0e7ff - PID: 04 (idx 000, nxt: 058)
77 058: 0e800-0ebff - PID: 04 (idx 001, nxt: 059)
78 059: 0ec00-0efff - PID: 04 (idx 002, nxt: 060)
79 0ede7: 0a
80 060: 0f000-0f3ff - PID: 04 (idx 003, nxt: 061)
81 061: 0f400-0f7ff - PID: 04 (idx 004, nxt: -01)
82 NOTE: Read file output/os_0 to verify your result
```

3.2 TEST 1

```
1 ----- OS TEST 1 -----
2 ./os os_1
3 Time slot 0
4   Loaded a process at input/proc/p0, PID: 1
5   CPU 3: Dispatched process 1
6 Time slot 1
7 Time slot 2
8   Loaded a process at input/proc/s3, PID: 2
9 Time slot 3
10  CPU 0: Dispatched process 2
11  CPU 3: Put process 1 to run queue
12  CPU 3: Dispatched process 1
13 Time slot 4
14  Loaded a process at input/proc/m1, PID: 3
15  CPU 2: Dispatched process 3
16 Time slot 5
17  CPU 0: Put process 2 to run queue
18  CPU 0: Dispatched process 2
19  CPU 3: Put process 1 to run queue
20  CPU 3: Dispatched process 1
21 Time slot 6
22  CPU 2: Put process 3 to run queue
23  CPU 2: Dispatched process 3
24  Loaded a process at input/proc/s2, PID: 4
25 Time slot 7
26  CPU 0: Put process 2 to run queue
27  CPU 0: Dispatched process 4
28  CPU 1: Dispatched process 2
29  Loaded a process at input/proc/m0, PID: 5
30  CPU 3: Put process 1 to run queue
31  CPU 3: Dispatched process 5
32 Time slot 8
33  CPU 2: Put process 3 to run queue
34  CPU 2: Dispatched process 1
35 Time slot 9
36  Loaded a process at input/proc/p1, PID: 6
37  CPU 1: Put process 2 to run queue
38  CPU 1: Dispatched process 3
39  CPU 0: Put process 4 to run queue
40  CPU 0: Dispatched process 6
41  CPU 3: Put process 5 to run queue
42  CPU 3: Dispatched process 4
43 Time slot 10
```



```
44 CPU 2: Put process 1 to run queue
45 CPU 2: Dispatched process 2
46 Time slot 11
47   Loaded a process at input/proc/s0, PID: 7
48 CPU 0: Put process 6 to run queue
49 CPU 0: Dispatched process 7
50 CPU 1: Put process 3 to run queue
51 CPU 1: Dispatched process 5
52 CPU 3: Put process 4 to run queue
53 CPU 3: Dispatched process 4
54 Time slot 12
55 CPU 2: Put process 2 to run queue
56 CPU 2: Dispatched process 1
57 Time slot 13
58 CPU 1: Put process 5 to run queue
59 CPU 1: Dispatched process 6
60 CPU 0: Put process 7 to run queue
61 CPU 0: Dispatched process 3
62 CPU 3: Put process 4 to run queue
63 CPU 3: Dispatched process 4
64 Time slot 14
65 CPU 2: Processed 1 has finished
66 CPU 2: Dispatched process 7
67 Time slot 15
68 CPU 3: Put process 4 to run queue
69 CPU 3: Dispatched process 2
70 CPU 1: Put process 6 to run queue
71 CPU 1: Dispatched process 5
72 CPU 0: Processed 3 has finished
73 CPU 0: Dispatched process 4
74 Time slot 16
75   Loaded a process at input/proc/s1, PID: 8
76 CPU 2: Put process 7 to run queue
77 CPU 2: Dispatched process 8
78 Time slot 17
79 CPU 0: Put process 4 to run queue
80 CPU 0: Dispatched process 6
81 CPU 1: Put process 5 to run queue
82 CPU 1: Dispatched process 4
83 CPU 3: Put process 2 to run queue
84 CPU 3: Dispatched process 7
85 Time slot 18
86 CPU 2: Put process 8 to run queue
87 CPU 2: Dispatched process 5
88 Time slot 19
89 CPU 3: Put process 7 to run queue
90 CPU 2: Processed 5 has finished
91 CPU 0: Put process 6 to run queue
92 CPU 1: Processed 4 has finished
93 CPU 1: Dispatched process 6
94 CPU 2: Dispatched process 7
95 CPU 3: Dispatched process 8
96 CPU 0: Dispatched process 2
97 Time slot 20
98 CPU 0: Processed 2 has finished
99 CPU 0 stopped
100 Time slot 21
101 CPU 3: Put process 8 to run queue
102 CPU 1: Put process 6 to run queue
103 CPU 1: Dispatched process 6
104 CPU 2: Put process 7 to run queue
105 CPU 2: Dispatched process 7
```



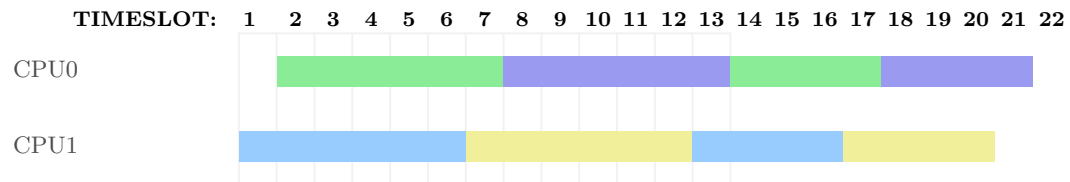
```
106 CPU 3: Dispatched process 8
107 Time slot 22
108 Time slot 23
109 CPU 3: Put process 8 to run queue
110 CPU 3: Dispatched process 8
111 CPU 2: Put process 7 to run queue
112 CPU 2: Dispatched process 7
113 CPU 1: Processed 6 has finished
114 CPU 1 stopped
115 Time slot 24
116 CPU 3: Processed 8 has finished
117 CPU 3 stopped
118 Time slot 25
119 CPU 2: Put process 7 to run queue
120 CPU 2: Dispatched process 7
121 Time slot 26
122 Time slot 27
123 CPU 2: Put process 7 to run queue
124 CPU 2: Dispatched process 7
125 Time slot 28
126 CPU 2: Processed 7 has finished
127 CPU 2 stopped
128
129 MEMORY CONTENT:
130 000: 00000-003ff - PID: 05 (idx 000, nxt: 001)
131 003e8: 15
132 001: 00400-007ff - PID: 05 (idx 001, nxt: -01)
133 002: 00800-00bff - PID: 05 (idx 000, nxt: 003)
134 003: 00c00-00fff - PID: 05 (idx 001, nxt: 004)
135 004: 01000-013ff - PID: 05 (idx 002, nxt: 005)
136 005: 01400-017ff - PID: 05 (idx 003, nxt: 006)
137 006: 01800-01bff - PID: 05 (idx 004, nxt: -01)
138 011: 02c00-02fff - PID: 06 (idx 000, nxt: 012)
139 012: 03000-033ff - PID: 06 (idx 001, nxt: 013)
140 013: 03400-037ff - PID: 06 (idx 002, nxt: 014)
141 014: 03800-03bff - PID: 06 (idx 003, nxt: -01)
142 021: 05400-057ff - PID: 01 (idx 000, nxt: -01)
143 05414: 64
144 024: 06000-063ff - PID: 05 (idx 000, nxt: 025)
145 06014: 66
146 025: 06400-067ff - PID: 05 (idx 001, nxt: -01)
147 031: 07c00-07fff - PID: 06 (idx 000, nxt: 032)
148 032: 08000-083ff - PID: 06 (idx 001, nxt: 033)
149 033: 08400-087ff - PID: 06 (idx 002, nxt: 034)
150 085e7: 0a
151 034: 08800-08bff - PID: 06 (idx 003, nxt: 035)
152 035: 08c00-08fff - PID: 06 (idx 004, nxt: -01)
153 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

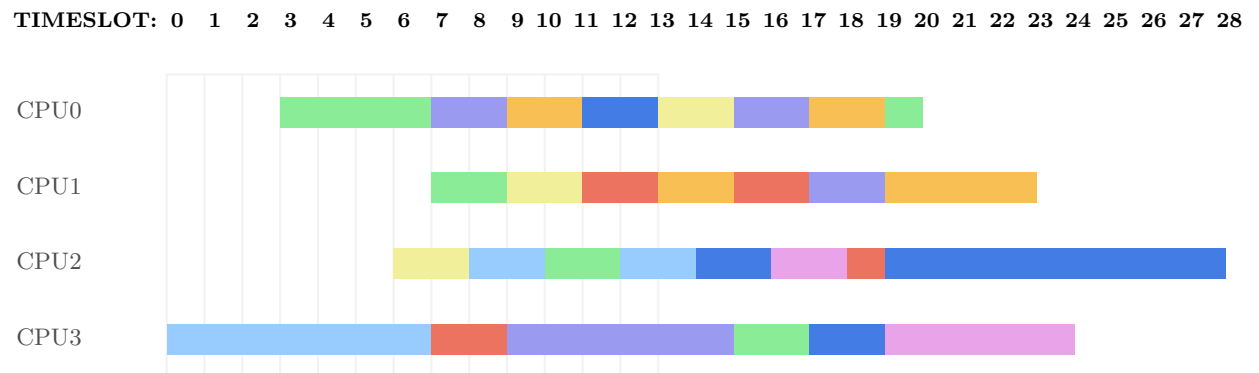


Gantt-chart

Test 0



Test 1



Note



