

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



Course: Operating Systems

Course Assignment

Simple Operating System

Advisor : Trần Trương Tuấn Phát
Student: Võ Thượng Bảo - 2210290
Đinh Công Minh - 2212027
Dương Hoàng Khôi - 2211672
Nguyễn Huy Hoàng - 2211091
Nguyễn Phan Duy Bảo - 2210244

HO CHI MINH CITY, APRIL 2024



Contents

Member list & Workload	2
1 Overview	3
2 Scheduler	3
2.1 Theoretical basis	3
2.2 Answer the question.	4
2.3 Implement	5
2.3.1 queue.c	5
2.3.2 sched.c	6
2.4 Execution Result	7
2.4.1 Testcase sched_0	7
2.4.2 Testcase sched_1	8
2.4.3 Testcase sched	10
3 Memory Management	12
3.1 Virtual Memory for each Process	12
3.2 Paging-based address translation scheme	14
3.3 Answer the question	15
3.4 Translation Lookaside Buffer (TLB)	17
3.4.1 Theoretical Basis	17
3.4.2 Answer the question	18
3.5 Structure of TLB	19
3.5.1 The values stored in the TLB	19
3.5.2 tlb-alloc	19
3.5.3 tlb-free	20
3.5.4 tlb-read	21
3.5.5 tlb-write	22
3.6 The process of accessing TLB	23
3.7 Implement	24
3.7.1 tlb_alloc function	24
3.7.2 tlb_free_data function	25
3.7.3 tlb_read function	25
3.7.4 tlb_write function	26
3.8 Execution Result	27
3.8.1 Testcase os_0_mlq_paging	27
3.8.2 Testcase os_1_mlq_paging_small_1K	30
4 Put it all together	38
4.1 Synchronization	38
4.1.1 Design	38
4.1.2 Implementation	38
4.1.2.a Without synchronization:	39
4.1.2.b With synchronization:	46
4.2 Answer the question.	54



Member list & Workload

No.	Fullname	Student ID	Problems	Percentage of work
1	Võ Thượng Bảo	2210290	Compile documents, write reports (Theoretical basis + problems 2.3).	20%
2	Đinh Công Minh	2212027	Compile documents, write reports (Theoretical basis + problem 2.2.1).	20%
3	Dương Hoàng Khôi	2211672	Problems 2.2.4,2.2.5	20%
4	Nguyễn Huy Hoàng	2211091	Problem 2.2.2,2.2.3	20%
5	Nguyễn Phan Duy Bảo	2210244	Problem 2.2.4,2.2.5	20%

1 Overview

The purpose of this assignment is to simulate a simple operating system, helping students understand the basic knowledge about scheduling, synchronization, and memory management. Figure 1 provides an overview of the structure of the operating system that we will implement. Fundamentally, the operating system will manage two virtual resources: CPUs and RAM, using two components:

- **Scheduler (and Dispatcher):** decide which process will be executed on which CPU.
- **Virtual Memory Engine (VME):** isolates the memory space of each process from one another. Physical RAM is shared among multiple processes, but each process is unaware of the others' existence. This is achieved by allowing each process to have its own virtual memory space, and the virtual memory manager will map and translate the virtual addresses provided by the processes into corresponding physical addresses.

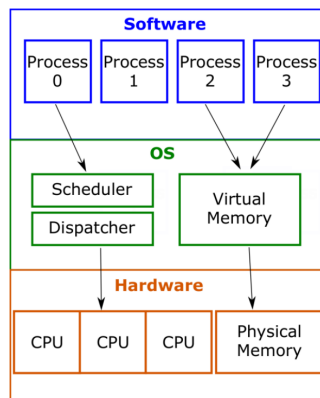


Figure 1: Overview of the modules in a simple operating system

Through the modules mentioned above, the operating system allows multiple user-created processes to share and utilize resources. Therefore, in this major assignment, we will proceed to implement the Scheduler/Dispatcher and the Virtual Memory Engine components.

2 Scheduler

2.1 Theoretical basis

In this assignment, we use the Multi-level Queue Scheduling (MLQ) algorithm to implement the Scheduler. Multi-level Queue Scheduling is an algorithmic structure for arranging tasks based on their priority levels and providing corresponding resources.

Features of the MLQ algorithm:

- The MLQ system contains MAX PRIO priority levels (Ready queue). The Ready queue is divided into multiple separate queues. Processes are placed into the ready queue to be allocated CPU time.

- In this assignment, we use the Round Robin algorithm to schedule CPU time, prioritizing from highest to lowest. The number of allocations for CPU usage of each ready queue in the list is a fixed formula based on priority, i.e., $\text{slot} = (\text{MAX PRIO} - \text{prio})$. When the allocation is exhausted, the system must switch resources to another process in the next queue.

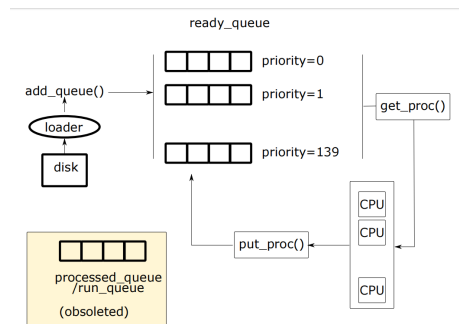


Figure 2: Operation of the Scheduler System

2.2 Answer the question.

Question: What are the advantages of using a Priority Queue over other scheduling algorithms?

Answer: The advantage of using a priority queue over other scheduling algorithms is that it allows for prioritization and efficient selection of processes based on their priorities. A priority queue is a data structure where elements are assigned priority levels, and the element with the highest priority is always selected first.

- **Scheduling based on priority:** A priority queue allows scheduling based on priorities, where processes with higher priorities are executed first. This enables the implementation of priority-based policies, such as preemptive scheduling, where a high-priority process can interrupt a lower-priority process that is currently executing.
- **Flexibility and Adaptability:** A priority queue can be easily updated and adjusted during runtime. When the priorities of processes change or new processes are created, the queue can be adjusted accordingly. This flexibility allows for efficient handling of dynamically changing priorities in real-time systems.
- **Efficient Selection:** With a priority queue, the process with the highest priority can be selected in constant time, regardless of the number of processes in the queue. This ensures efficient selection for the next process to be executed, reducing the time complexity of the scheduling algorithm.
- **Customizable Priorities:** A priority queue provides flexibility to assign and adjust priorities based on specific criteria or policies. The priority of a process can be determined by factors such as process importance, deadline constraints, resource requirements, or specific metrics of any application.
- **Fairness and Responsiveness:** By assigning different priorities to processes, a priority queue can ensure fairness and responsiveness in scheduling. Processes with higher priorities receive more CPU time, leading to better responsiveness for important tasks or operations.

requiring specific time constraints. It is important to emphasize that while priority queues offer scheduling advantages based on priorities, they may not be suitable for all situations. The choice of scheduling algorithm depends on the specific requirements of the operating system or application, considering factors such as fairness, throughput, response time, and system load.

2.3 Implement

2.3.1 queue.c

enqueue function: Add process to the queue.

```
1 void enqueue(struct queue_t * q, struct pcb_t * proc) {
2     /* TODO: put a new process to queue [q] */
3     int sizeOfQueue = q->size;
4     if(sizeOfQueue < MAX_QUEUE_SIZE && sizeOfQueue >= 0)
5     {
6         q->proc[sizeOfQueue] = proc;
7         q->size = sizeOfQueue + 1;
8     }
9 }
10
```

dequeue function: Function returns the highest priority process in the queue and removes it from the queue.

```
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     int sizeOfQueue = q->size;
6     if( sizeOfQueue == 1 )
7     {
8         struct pcb_t* return_proc = q->proc[0];
9         q->proc[0] = NULL;
10        q->size = 0;
11        return return_proc;
12    }
13    else if( sizeOfQueue > 1 && sizeOfQueue <= MAX_QUEUE_SIZE )
14    {
15        struct pcb_t* return_proc = q->proc[0]; // process will be returned
16        uint32_t highest_prior = q->proc[0]->priority;
17        int position = 0;
18        int i;
19        for(i = 0 ; i < sizeOfQueue ; i++)
20        {
21            if( highest_prior < q->proc[i]->priority )
22            {
23                highest_prior = q->proc[i]->priority;
24                position = i;
25            }
26        }
27        return_proc = q->proc[position];
28        if(position == (sizeOfQueue - 1)) // highest at the end of queue
29        {
30            q->proc[position] = NULL;
31            q->size = sizeOfQueue - 1;
32            return return_proc;
33        }
34        else
```

```
35     {
36         for(i = position+1; i < sizeofQueue; i++)
37         {
38             q->proc[i-1] = q->proc[i];
39         }
40         q->proc[sizeofQueue - 1] = NULL;
41         q->size = sizeofQueue - 1;
42         return return_proc;
43     }
44 }
45 return NULL;
46 }
47
```

2.3.2 sched.c

get-mlq-proc function

```
1     struct pcb_t * get_mlq_proc(void) {
2     struct pcb_t *proc = NULL;
3     /*TODO: get a process from PRIORITY [ready_queue].
4      * Remember to use lock to protect the queue.
5      */
6     pthread_mutex_lock(&queue_lock);
7     int i=0;
8     int isEmpty =0;
9     while(1)
10    {
11        if (empty(&mlq_ready_queue[i]) || slot[i] == 0)
12        {
13            if(i >= MAX_PRIO -1){
14                if(isEmpty == 1){
15                    break;
16                }
17                slot[i] = MAX_PRIO - i;
18                i = -1;
19                isEmpty++;
20                i++;
21            }
22            slot[i] = MAX_PRIO - i;
23            i++;
24        }
25        else{
26            proc = dequeue(&mlq_ready_queue[i]);
27            slot[i]--;
28            break;
29        }
30    }
31    pthread_mutex_unlock(&queue_lock);
32    return proc;
33    }
34
```

get-proc function

```
1     struct pcb_t *get_proc(void)
2     {
3         struct pcb_t *proc = NULL;
4         /*TODO: get a process from [ready_queue].
5          * Remember to use lock to protect the queue.
6          */

```

```
7 pthread_mutex_lock(&queue_lock);
8 if (empty(&ready_queue))
9 {
10     while (!empty(run_queue))
11     {
12         enqueue(&ready_queue, dequeue(&run_queue));
13     }
14 }
15 proc = dequeue(&ready_queue);
16 pthread_mutex_unlock(&queue_lock);
17
18 return proc;
19 }
20
```

2.4 Execution Result

2.4.1 Testcase sched_0

Input:

```
1 2 1 2
2 2048 16777216 0 0 0
3 0 s0 4
4 4 s1 0
5
```

Output:

```
1 Time slot 0
2 ld_routine
3     Loaded a process at input/proc/s0, PID: 1 PRI0: 4
4 Time slot 1
5     CPU 0: Dispatched process 1
6 Time slot 2
7 Time slot 3
8     CPU 0: Put process 1 to run queue
9     CPU 0: Dispatched process 1
10 Time slot 4
11     Loaded a process at input/proc/s1, PID: 2 PRI0: 0
12 Time slot 5
13     CPU 0: Put process 1 to run queue
14     CPU 0: Dispatched process 2
15 Time slot 6
16 Time slot 7
17     CPU 0: Put process 2 to run queue
18     CPU 0: Dispatched process 2
19 Time slot 8
20 Time slot 9
21     CPU 0: Put process 2 to run queue
22     CPU 0: Dispatched process 2
23 Time slot 10
24 Time slot 11
25     CPU 0: Put process 2 to run queue
26     CPU 0: Dispatched process 2
27 Time slot 12
28     CPU 0: Processed 2 has finished
29     CPU 0: Dispatched process 1
30 Time slot 13
31 Time slot 14
32     CPU 0: Put process 1 to run queue
```




```
33 CPU 0: Dispatched process 1
34 Time slot 15
35 Time slot 16
36 CPU 0: Put process 1 to run queue
37 CPU 0: Dispatched process 1
38 Time slot 17
39 Time slot 18
40 CPU 0: Put process 1 to run queue
41 CPU 0: Dispatched process 1
42 Time slot 19
43 Time slot 20
44 CPU 0: Put process 1 to run queue
45 CPU 0: Dispatched process 1
46 Time slot 21
47 Time slot 22
48 CPU 0: Put process 1 to run queue
49 CPU 0: Dispatched process 1
50 Time slot 23
51 CPU 0: Processed 1 has finished
52 CPU 0 stopped
53
```

Grant chart:

process	P1					P2								P1							
Time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

process	P1																	
Time	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38

2.4.2 Testcase sched_1

Input:

```
1 2 1 4
2 2048 16777216 0 0 0
3 0 s0 4
4 4 s1 0
5 6 s2 0
6 7 s3 0
7
```

Output:

```
1 Time slot 0
2 ld_routine
3 Loaded a process at input/proc/s0, PID: 1 PRI0: 4
4 Time slot 1
5 CPU 0: Dispatched process 1
6 Time slot 2
7 Time slot 3
8 CPU 0: Put process 1 to run queue
9 CPU 0: Dispatched process 1
10 Time slot 4
11 Loaded a process at input/proc/s1, PID: 2 PRI0: 0
12 Time slot 5
13 CPU 0: Put process 1 to run queue
14 CPU 0: Dispatched process 2
15 Time slot 6
16 Loaded a process at input/proc/s2, PID: 3 PRI0: 0
17 Time slot 7
```



```
18      CPU 0: Put process 2 to run queue
19      CPU 0: Dispatched process 3
20      Loaded a process at input/proc/s3, PID: 4 PRI0: 0
21 Time slot 8
22 Time slot 9
23      CPU 0: Put process 3 to run queue
24      CPU 0: Dispatched process 2
25 Time slot 10
26 Time slot 11
27      CPU 0: Put process 2 to run queue
28      CPU 0: Dispatched process 3
29 Time slot 12
30 Time slot 13
31      CPU 0: Put process 3 to run queue
32      CPU 0: Dispatched process 2
33 Time slot 14
34 Time slot 15
35      CPU 0: Put process 2 to run queue
36      CPU 0: Dispatched process 3
37 Time slot 16
38 Time slot 17
39      CPU 0: Put process 3 to run queue
40      CPU 0: Dispatched process 2
41 Time slot 18
42      CPU 0: Processed 2 has finished
43      CPU 0: Dispatched process 3
44 Time slot 19
45 Time slot 20
46      CPU 0: Put process 3 to run queue
47      CPU 0: Dispatched process 3
48 Time slot 21
49 Time slot 22
50      CPU 0: Put process 3 to run queue
51      CPU 0: Dispatched process 3
52 Time slot 23
53 Time slot 24
54      CPU 0: Processed 3 has finished
55      CPU 0: Dispatched process 4
56 Time slot 25
57 Time slot 26
58      CPU 0: Put process 4 to run queue
59      CPU 0: Dispatched process 4
60 Time slot 27
61 Time slot 28
62      CPU 0: Put process 4 to run queue
63      CPU 0: Dispatched process 4
64 Time slot 29
65 Time slot 30
66      CPU 0: Put process 4 to run queue
67      CPU 0: Dispatched process 4
68 Time slot 31
69 Time slot 32
70      CPU 0: Put process 4 to run queue
71      CPU 0: Dispatched process 4
72 Time slot 33
73 Time slot 34
74      CPU 0: Put process 4 to run queue
75      CPU 0: Dispatched process 4
76 Time slot 35
77      CPU 0: Processed 4 has finished
78      CPU 0: Dispatched process 1
79 Time slot 36
```



```
80 Time slot 37
81     CPU 0: Put process 1 to run queue
82     CPU 0: Dispatched process 1
83 Time slot 38
84 Time slot 39
85     CPU 0: Put process 1 to run queue
86     CPU 0: Dispatched process 1
87 Time slot 40
88 Time slot 41
89     CPU 0: Put process 1 to run queue
90     CPU 0: Dispatched process 1
91 Time slot 42
92 Time slot 43
93     CPU 0: Put process 1 to run queue
94     CPU 0: Dispatched process 1
95 Time slot 44
96 Time slot 45
97     CPU 0: Put process 1 to run queue
98     CPU 0: Dispatched process 1
99 Time slot 46
100    CPU 0: Processed 1 has finished
101    CPU 0 stopped
102
```

Grant chart:

process		P1				P2		P3		P2		P3		P2		P3		P2		P3	
Time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

process	P3		P4												P1					
Time	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

process	P1																			
Time	41	42	43	44	45	46	27	28	29	30	31	32	33	34	35	36	37	38	39	40

2.4.3 Testcase sched

Input:

```
1 4 2 3
2 2048 16777216 0 0 0
3 0 p1s 1
4 1 p2s 0
5 2 p3s 0
6
```

Output:

```
1 Time slot 0
2 ld_routine
3     Loaded a process at input/proc/p1s, PID: 1 PRI0: 1
4 Time slot 1
5     CPU 1: Dispatched process 1
6     Loaded a process at input/proc/p2s, PID: 2 PRI0: 0
7 Time slot 2
```



```
8      CPU 0: Dispatched process 2
9      Loaded a process at input/proc/p3s, PID: 3 PRI0: 0
10     Time slot 3
11     Time slot 4
12      CPU 1: Put process 1 to run queue
13      CPU 1: Dispatched process 3
14     Time slot 5
15     Time slot 6
16      CPU 0: Put process 2 to run queue
17      CPU 0: Dispatched process 2
18     Time slot 7
19     Time slot 8
20      CPU 1: Put process 3 to run queue
21      CPU 1: Dispatched process 3
22     Time slot 9
23     Time slot 10
24      CPU 0: Put process 2 to run queue
25      CPU 0: Dispatched process 2
26     Time slot 11
27     Time slot 12
28     Time slot 13
29      CPU 1: Put process 3 to run queue
30      CPU 1: Dispatched process 3
31     Time slot 14
32      CPU 0: Processed 2 has finished
33      CPU 0: Dispatched process 1
34     Time slot 15
35     Time slot 16
36      CPU 1: Processed 3 has finished
37      CPU 1 stopped
38     Time slot 17
39     Time slot 18
40      CPU 0: Put process 1 to run queue
41      CPU 0: Dispatched process 1
42     Time slot 19
43     Time slot 20
44      CPU 0: Processed 1 has finished
45      CPU 0 stopped
46
```

Grant chart:

Process/Time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1(PRIO: 1)																							
2(PRIO: 0)																							
3(PRIO: 0)																							

CPU 1	
CPU 0	

3 Memory Management

3.1 Virtual Memory for each Process

During the execution of processes, sharing a page table can lead to various issues and make control more challenging. Therefore, each process in this BTL will have its own page table and virtual memory regions.

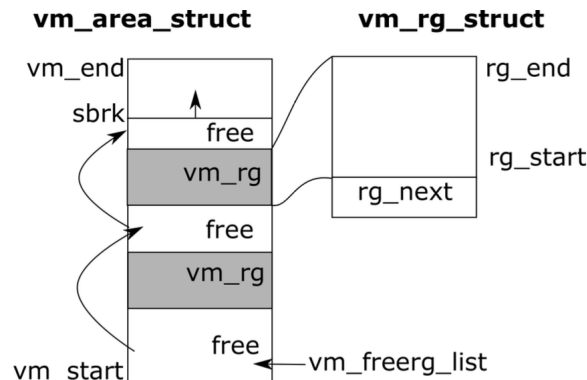


Figure 3: The structure of vm area and region

vm_area: Each memory region extends continuously within the range [vm start, vm end]. Although the space spans the entire range, the actual usable region may be limited by the top pointer into sbrk. Within the region between vm start and sbrk, there are multiple areas identified by the vm rg struct structure, and the free spaces are tracked by the vm freerg list.

vm_rg: These regions are actually considered as variables in the program's source code that can be human-readable. Since the current reality is beyond the scope mentioned, we simply generalize the concept of namespace within the index scope. We temporarily imagine these regions as a set of limited quantity regions. We manage them using an array symrgtbl[PAGING MAX SYMTBL SZ]. The size of the array is fixed by a constant, PAGING MAX SYMTBL SZ, which specifies the number of variables allowed in each program. Memory mapping: Memory regions are separate within an adjacent memory region. In each memory mapping structure, multiple memory regions are indicated by struct vm area struct *mmap list. The next important field is pgd, which is the page directory table containing all the page table entries. Each entry is a mapping between page numbers and frame numbers in the page management system. Symrgtbl is a simple implementation of a symbol table. Other fields are mainly used to track specific user activities, for example, syscalls, page fifo (for reference).

Each process also has a page address (or logical address) provided by the CPU (referred to in the assignment as the CPU address) to access a specific memory location, as depicted in the diagram below, specifically including:

- **Page number (p):** the index of a page table storing the base address of each process's memory area (page) in physical memory.
- **Page offset (d):** combined with the base address to determine the physical memory address sent to the Memory Management Unit.

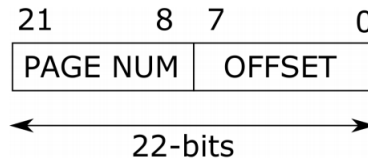


Figure 4: CPU Address

Physical Memory for each Process Physical Memory, also known as RAM (Random Access Memory), is the main memory of a computer, where data and programs currently used by the CPU are stored. It is a semiconductor memory, which means it can quickly write and erase data.

Memory hardware is a crucial part of the computer system, comprising primary memory (RAM) and secondary memory (SWAP). The RAM device is accessed directly from the CPU's address bus, allowing for fast read and write operations. SWAP, on the other hand, acts as a secondary storage device used to store data that cannot fit in RAM. SWAP must transfer data to the main memory before operation, making it slower compared to RAM.

Both RAM and SWAP can be implemented on the same physical hardware but are used differently. The system can have various configurations that affect how memory is accessed and organized. Settings such as random, sequential, and storage capacity can be customized to meet specific needs.

RAM: it often used as the main memory due to its capability for fast direct access from the CPU. It allows for read and write operations based on CPU commands. This memory is typically more expensive, hence it is kept small yet very fast. It is depicted as a single physical device in Figure 1.

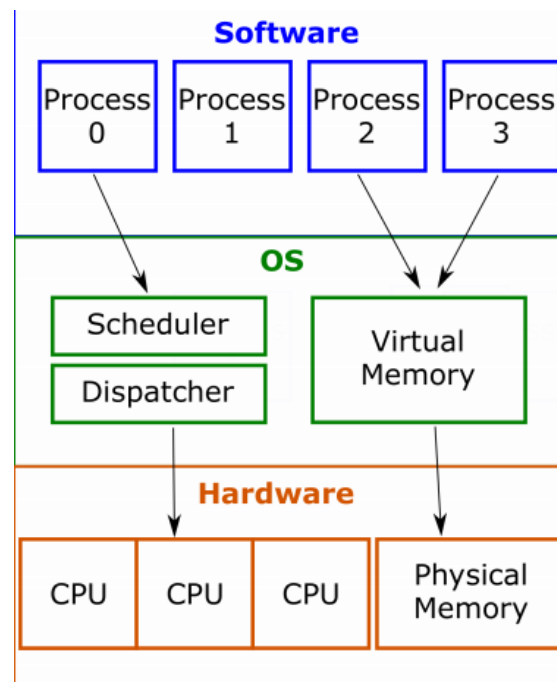


Figure 5: The general view of key modules in this assignment

SWAP: functions as secondary memory, designed to manage surplus data when RAM is full. Since it does not have direct CPU access, data must be transferred to the main memory before operations, making it slower but offering additional memory capacity at a lower cost. Systems can have multiple SWAP devices to increase storage capacity.

3.2 Paging-based address translation scheme

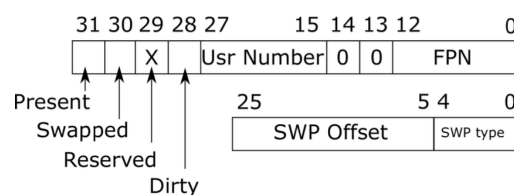


Figure 6: The content of page table entry

Page table: This structure allows a process in user space to determine which physical frame each virtual page is mapped to. It contains a 32-bit value for each virtual page, including the following data:

```
* Bits 0-12 page frame number (FPN) if present
* Bits 13-14 zero if present
* Bits 15-27 user-defined numbering if present
* Bits 0-4 swap type if swapped
* Bits 5-25 swap offset if swapped
* Bit 28 dirty
* Bits 29 reserved
* Bit 30 swapped
* Bit 31 presented
```

Memory swapping: When the data in RAM is full, if a process requests an allocation, this cannot be performed. However, with virtual memory and the assistance of Memory swapping, this can be achieved. Swapping can help move the contents of a physical frame between MEMRAM and MEMSWAP. Swapping is the mechanism of copying the content of a frame from the outside into the main RAM memory. Conversely, swapping out attempts to move the content of a frame in MEMRAM to MEMSWAP. In a typical scenario, swapping helps us obtain free RAM frames because the size of the SWAP device is usually large enough. However, in practice, this process takes much longer than reading from RAM, so with small RAMs, frequent swapping makes program execution very slow.

Basic memory operations in paging-based system:

- **ALLOC:** In most cases, providing an Alloc corresponds to an available region. If there is no suitable space, we need to increase the sbrk limit, and since it has never been used, we may need to provide some physical frames and then map them using the Page Table.
- **FREE:** The storage space associated with the region ID. Since we cannot reclaim allocated physical frames, this may lead to memory holes. We only keep the reclaimed storage space in the free list for the next allocation request.
- **READ/WRITE:** The page request must be present in main memory. The most resource-intensive step is page swapping. If that page is in the MEMSWAP device, it needs to be brought back to the MEMRAM device (swapped in), and if there is a shortage of space, we need to swap out some pages to the MEMSWAP device to make room.

3.3 Answer the question

Question: What is the advantage and disadvantage of segmentation with paging?

Answer: Segmented Paging is a popular memory management method in operating systems. It combines two memory management concepts: segmentation and paging, to leverage the benefits of both approaches. Below are some advantages and disadvantages of this mechanism.

Advantages:

- **Flexibility:** By combining segmentation and paging, this mechanism allows for more flexible memory management designs. Segments can represent different logical areas of a program, while paging enables more efficient sharing of physical memory.
- **Security Management and Resource Sharing:** Segmentation provides better security and resource sharing capabilities. Each segment can be given its own access permissions,

helping protect against unauthorized access. Segments can also be shared among different processes.

- **Reduction of Internal Fragmentation:** With paging, larger segments are divided into smaller pages, maximizing the use of memory space. This reduces internal fragmentation compared to traditional segmentation.
- **Flexibility in Memory Management:** Segmented paging allows for more flexible memory allocation. It permits segments of varying sizes and can be expanded as needed, without the need to move other segments.

Disadvantages:

- **Higher Management Cost:** The segmented paging mechanism is more complex compared to using only segmentation or paging. This leads to higher management and computation costs, as it requires maintaining both segment tables and page tables.
- **Reduced Performance:** When accessing memory, the operating system must perform lookups in both the segment table and the page table, leading to increased lookup costs. This can reduce memory access performance compared to simpler memory management mechanisms.
- **Difficulty in Predicting Required Memory Size:** Segmented paging can make it challenging to predict the necessary memory size due to the combination of two methods. This can complicate memory optimization efforts.

Conclusion: The combined segmentation and paging mechanism is a flexible and powerful memory management method, but it also has drawbacks related to management overhead and performance. The choice between memory management mechanisms depends on the specific requirements of the system and applications.

Question: What would happen if we divide the address into more than 2 levels in a paging memory management system?

Answer: If we divide the address into more than 2 levels in the paging memory management system, the RAM and SWAP devices, dividing the address into more levels may have some additional considerations: If we divide the address into more than 2 levels in the paging memory management system, it allows for a larger address space to be represented. Each additional level of paging provides more bits for addressing, enabling a larger number of virtual and physical pages.

Dividing the address into more levels increases the flexibility and scalability of the memory management system. It allows for a finer-grained mapping of virtual pages to physical pages, which can improve memory utilization and reduce fragmentation. It also enables the system to handle larger amounts of memory efficiently.

However, increasing the number of levels in the paging system also introduces additional complexity and overhead. Each level requires additional memory for storing page tables or page directory entries, and it adds extra levels of indirection during address translation, which can impact performance.

Overall, dividing the address into more than 2 levels in the paging memory management system can provide benefits in terms of addressing capacity and memory utilization, but it also comes with increased complexity and potential performance trade-offs. The decision to use more levels

Question: In this simple operating system, implementing a design with multiple memory seg-

ments or memory regions in the source code declaration offers what benefits?

Answer: Designing multiple memory segments in an operating system has several important benefits, including:

- **Memory Organization :** By dividing the memory into multiple segments, the design provides a structured and organized layout for different types of data and code. Each segment can be dedicated to a specific purpose, such as the text segment for storing program instructions, the data segment for storing initialized data, and the heap segment for dynamic memory allocation. This organization improves the overall management and accessibility of different types of memory.
- **Memory Protection:** The use of multiple memory segments allows for memory protection and isolation. Each segment can have its own access permissions, such as read-only, read-write, or execute-only. This ensures that processes or segments cannot unintentionally modify or access memory areas that they are not supposed to, enhancing system security and stability.
- **Virtual Memory Mapping:** The virtual memory engine in the simple operating system maps virtual addresses provided by processes to corresponding physical addresses. The design of multiple memory segments facilitates this mapping process by providing clear boundaries and mappings between virtual and physical memory. This enables efficient address translation and memory management.
- **Modularity and Flexibility:** The design of multiple memory segments enables modularity and flexibility in managing memory. It provides a mechanism to allocate and deallocate memory in a granular manner, allowing efficient memory utilization and avoiding fragmentation. It also allows for dynamic resizing of specific segments based on the needs of processes or applications.
- **Code Readability and Maintainability:** By explicitly declaring multiple memory segments in the source code, the design improves code readability and maintainability. It provides a clear understanding of the memory layout and usage, making it easier to debug and modify the code in the future.

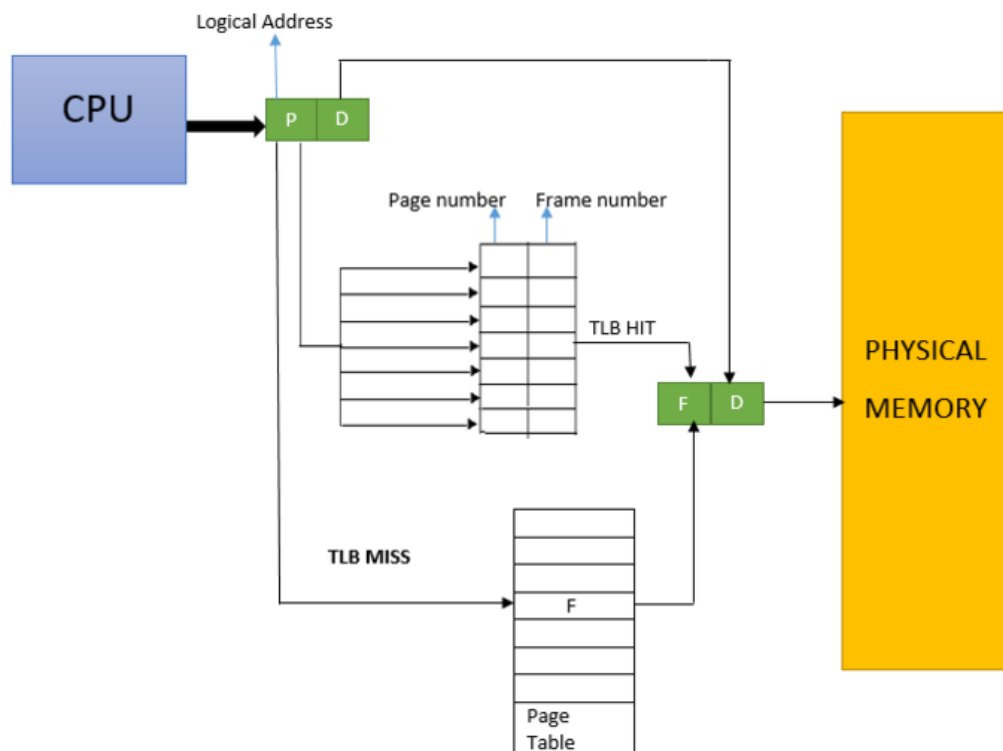
Conclusion: The design of multiple memory segments in the simple operating system enhances memory organization, protection, modularity, and code maintainability, contributing to the overall efficiency and reliability of the operating system.

3.4 Translation Lookaside Buffer (TLB)

3.4.1 Theoretical Basis

Translation Lookaside Buffer (TLB), also known as the translation cache, serves as a cache memory for the page table. Due to the high cost of producing cache memory, TLBs have a small capacity.

Similarly to other cache memories, each entry of the TLB consists of components such as a valid bit, a tag, and a data field. The data field in TLB entries contains the cached physical page numbers retrieved from the page table. Additionally, the entries in the page table also contain additional information. Therefore, the TLB blocks also contain this information either within the data field or in a separate field, but its significance is similar to that of the data field.



3.4.2 Answer the question

Question: What will happen if the multi-core system has each CPU core can be run in a different context, and each core has its own MMU and its part of the core (the TLB)? In modern CPU, 2-level TLBs are common now, what is the impact of these new memory hardware configurations to our translation schemes?

Answer: The hardware configuration of a new memory system with a two-level TLB on modern CPUs impacts our translation plans as follows:

- **Performance Improvement:** A two-level TLB can enhance translation performance by maintaining a small cache memory (L1 TLB) close to the processor and a larger cache (L2 TLB) storing more TLB entries. This reduces TLB lookup time and improves memory access performance.
- **Increased Capacity:** With a two-level TLB, we can have a larger TLB capacity, reducing the TLB miss rate and increasing overall system performance.
- **More Efficient Cache Management:** Two-level TLB allows for enhanced cache management by dividing TLB entries into smaller groups, reducing the lookup and update overhead.
- **Support for Complex Translation:** Two-level TLB can support more complex translation modes, including hierarchical mappings and enhanced memory protection, effectively supporting more complex security models and virtualization.

- **Reduced Memory Access Latency:** By reducing the TLB miss rate and enhancing TLB lookup performance, a two-level TLB helps reduce memory access latency, improving overall system performance.

3.5 Structure of TLB

3.5.1 The values stored in the TLB

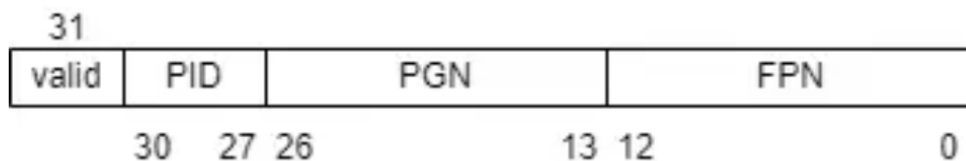


Figure 7: The position of bits in a TLB entry

Bit 31 (Most Significant Bit): Valid bit, indicating whether the entry is valid or not.

Bits 27-30: Process ID (PID) of the corresponding process.

Bits 13-26: Page number, identifying the virtual page within the process's address space.

Bits 0-12 (Least Significant Bits): Frame number, indicating the physical frame where the corresponding page is located in physical memory.

3.5.2 tlb-alloc

This is one of the most important functions of the TLB. This function allocates memory for a process, and then updates the TLB with the new memory region. The memory allocation process is essentially similar to regular memory allocation, however, after memory is allocated, it is combined with some other values such as: process ID (pid), page number (pgn), validity (valid), etc., and stored in the TLB for convenient use. The process of searching for the position to update will be discussed in the subsequent section

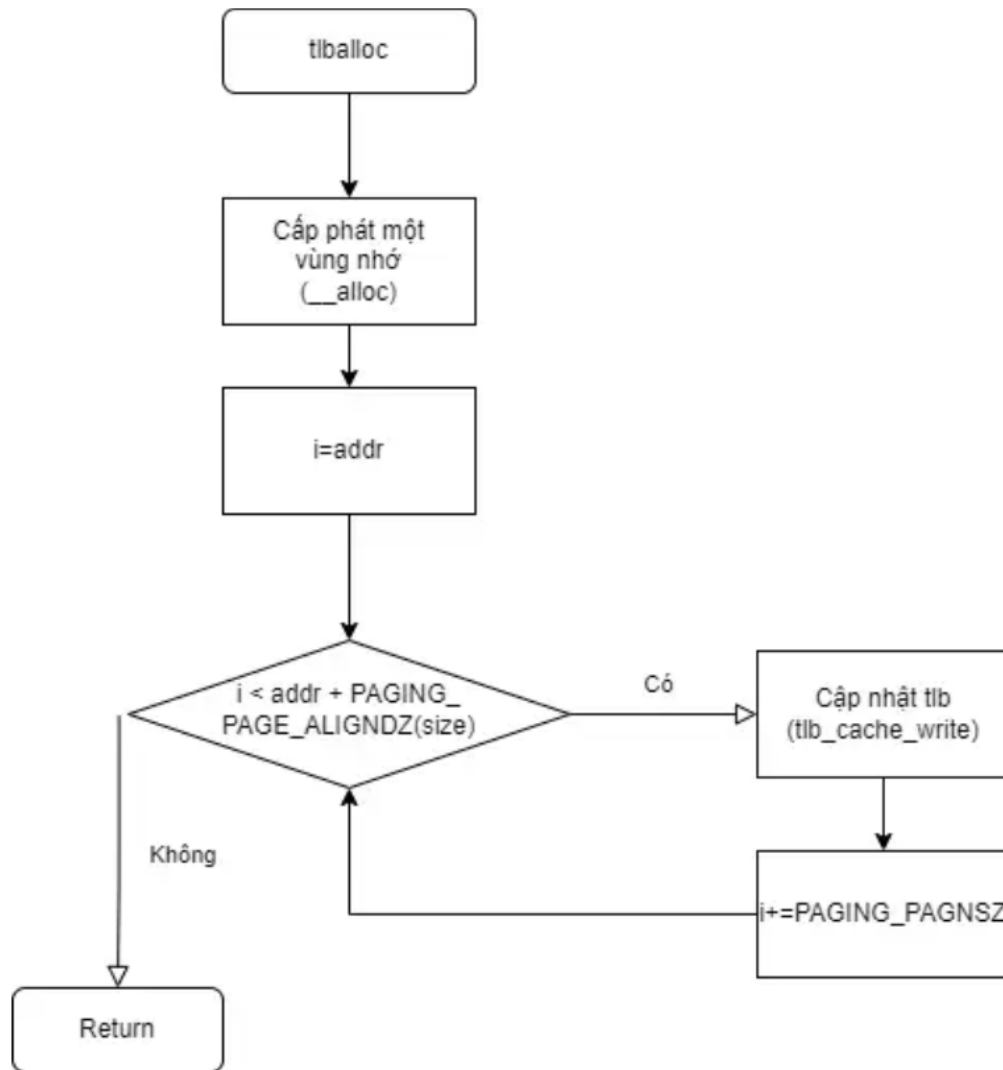


Figure 8: The TLB allocation process

3.5.3 tlb-free

This function is responsible for releasing the allocated memory stored in the register. This memory release operation is accompanied by releasing the corresponding TLB entry if it exists. After the data is released, any associated TLB entries are also deallocated.

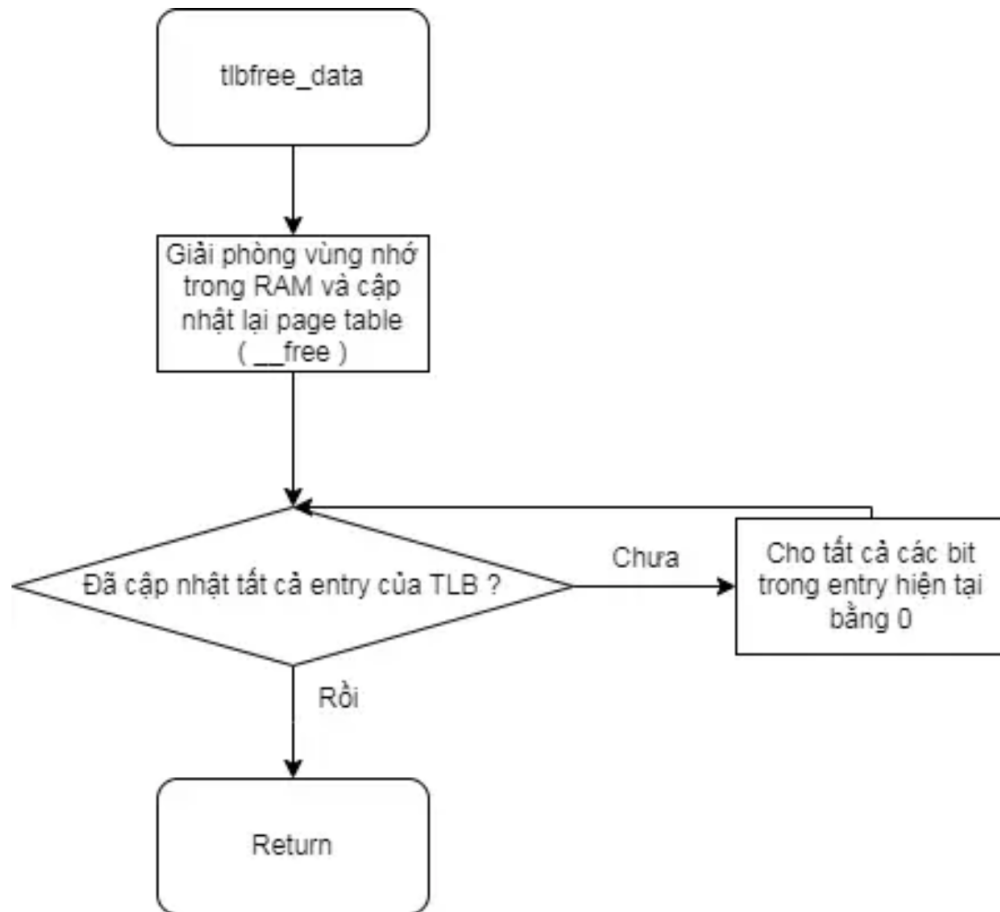


Figure 9: The TLB free process

3.5.4 tlb-read

This function is used to search for the value to be read in the TLB. If the value exists, it reads directly from memory. If not, it searches from the page table and updates the TLB accordingly.

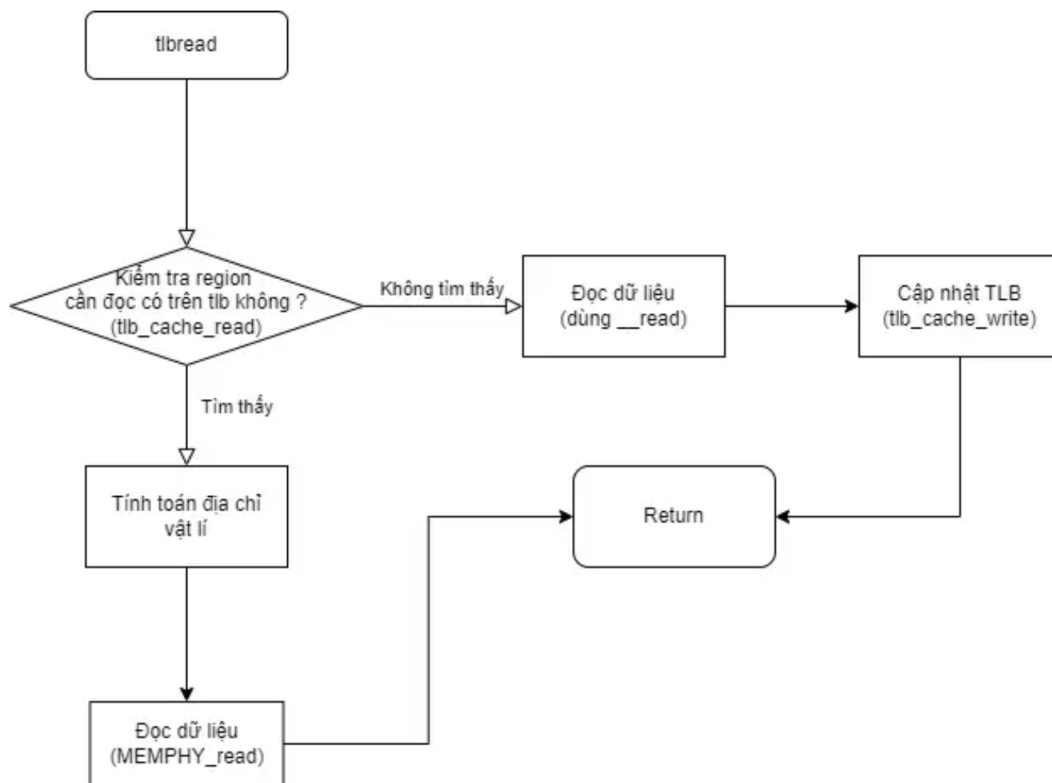


Figure 10: The TLB read process

3.5.5 tlb-write

The operation of TLB read works similarly, except instead of reading, it writes data.

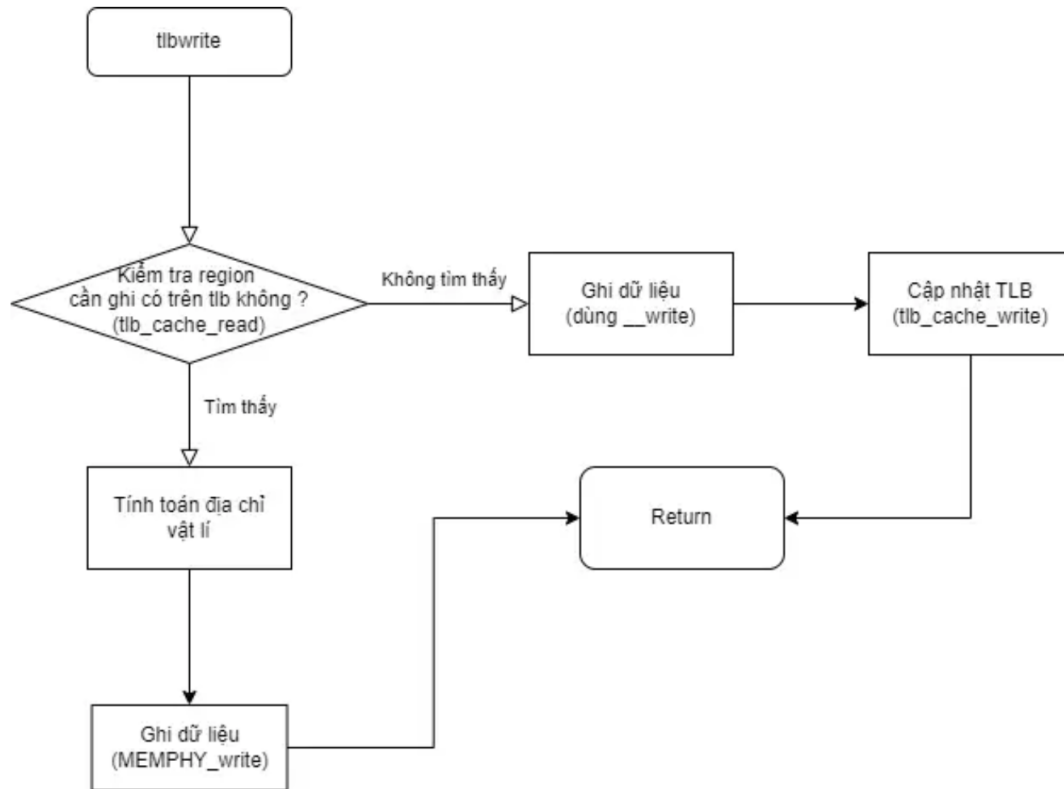


Figure 11: The TLB write process

3.6 The process of accessing TLB

In this assignment, my team implements direct mapping to access the TLB. Here, the team uses direct-mapping from the PID and PGN to map to the physical address of the TLB.

- **Advantages:** Using direct mapping makes accessing the TLB fast and straightforward.
- **Disadvantages:** It doesn't fully utilize the TLB, and there may be cases where processes occupy the same region, leading to frequent collisions that reduce TLB efficiency and increase data access time. For complex memory access models, direct mapped TLB may cause conflicts and limit access to virtual memory pages. Some virtual memory pages may not be effectively used in a direct mapped TLB, resulting in the TLB not utilizing its full storage capacity.

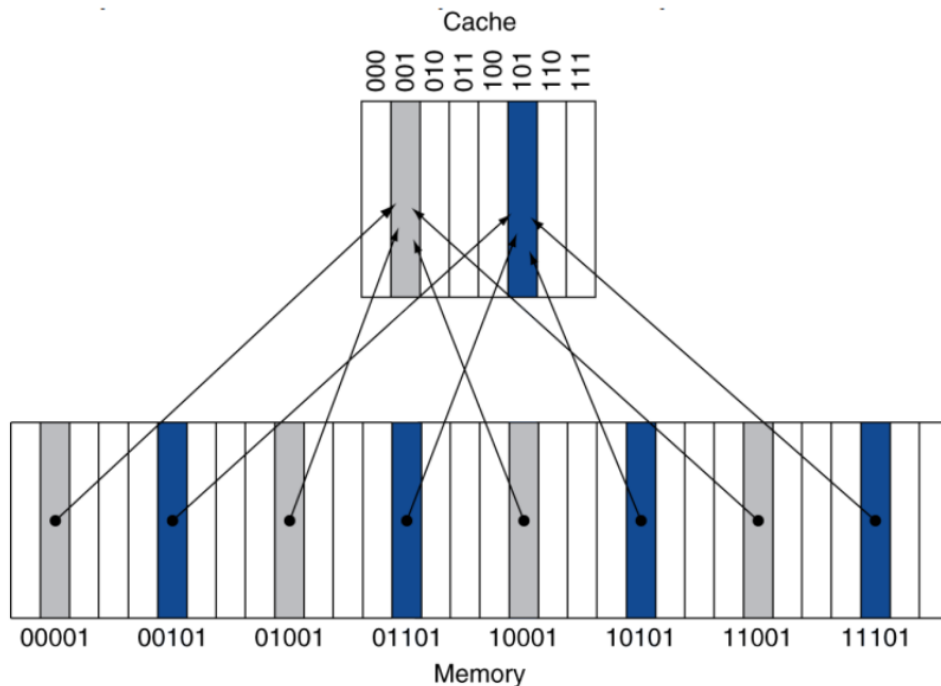


Figure 12: Direct mapping

Update strategy for the TLB :

- When the TLB is not full (i.e., there exists an entry with valid = 0), updating it involves finding the first available empty frame and writing the value into it.
- When the TLB is full, updating an entry is performed by overwriting the new value onto the current slot.

3.7 Implement

3.7.1 tlb_alloc function

```

1  int tlballoc(struct pcb_t *proc, uint32_t size, uint32_t reg_index)
2  {
3      // alloc size region
4      int addr, val;
5      /* By default using vmaid = 0 */
6      val = __alloc(proc, 0, reg_index, size, &addr);
7      /* TODO update TLB CACHED frame num of the new allocated page(s)*/
8      /* by using tlb_cache_read()/tlb_cache_write()*/
9      for (int i = addr; i < addr + PAGING_PAGE_ALIGNSZ(size); i += PAGING_PAGESZ)
10     {
11         tlb_cache_write(proc->tlb, proc->pid,
12                         PAGING_PGN(i), // dung
13                         PAGING_FPN_v2(proc->mm->pgd[PAGING_PGN(i)]));
14     }
15     return val;
16 }
17

```

3.7.2 tlb_free_data function

```
1  int tlbfree_data(struct pcb_t *proc, uint32_t reg_index)
2  {
3      __free(proc, 0, reg_index);
4      /* TODO update TLB CACHED frame num of freed page(s)*/
5      /* by using tlb_cache_read()/tlb_cache_write()*/
6      struct vm_rg_struct temp = proc->mm->symrgtbl[reg_index]; // co dung ko?
7      for (int i = temp.rg_start; i < temp.rg_end; i += PAGING_PAGESZ)
8      {
9          int pgn = PAGING_PGN(i);
10         if (proc->tlb->tlbd[pgn % proc->tlb->maxsz] != 0)
11         {
12             proc->tlb->tlbd[pgn % proc->tlb->maxsz] = 0;
13         }
14     }
15     return 0;
16 }
17
```

3.7.3 tlb_read function

```
1  int tlbread(struct pcb_t *proc, uint32_t source,
2              uint32_t offset, uint32_t destination)
3  {
4
5      /* TODO retrieve TLB CACHED frame num of accessing page(s)*/
6      /* by using tlb_cache_read()/tlb_cache_write()*/
7      /* frmnum is return value of tlb_cache_read/write value*/
8      BYTE data;
9      int pgnum = PAGING_PGN(proc->mm->symrgtbl[source].rg_start); // ko cong offset
10     uint16_t frnum;
11     int return_value = tlb_cache_read(proc->tlb, proc->pid, pgnum, &frnum);
12     proc->tlb->total_access++;
13     #ifdef IODUMP
14     if (return_value >= 0)
15     {
16         printf("TLB hit at read region=%d offset=%d\n",
17             source, offset); //?offset
18     }
19     else
20     {
21         printf("TLB miss at read region=%d offset=%d\n",
22             source, offset);
23     }
24     #ifdef PAGETBL_DUMP
25     print_pgtbl(proc, 0, -1); // print max TBL
26     #endif
27     MEMPHY_dump(proc->mram);
28     #endif
29     int val;
30     {
31         if (return_value >= 0)
32         {
33             proc->tlb->hit++;
34             // MEMPHY read
35             int phyaddr = (frnum << PAGING_ADDR_FPN_LOBIT) + offset;
36             val = MEMPHY_read(proc->mram, phyaddr, &data);
37         }
38         else
39         {

```

```
40     //__read
41     val = __read(proc, 0, source, offset, &data);
42
43     tlb_cache_write(proc->tlb, proc->pid, pgnum, PAGING_FPN_v2(proc->mm->pgd[
44         pgnum]));
45     }
46     }
47     destination = (uint32_t)data;
48     /* TODO update TLB CACHED with frame num of recent accessing page(s)*/
49     /* by using tlb_cache_read()/tlb_cache_write()*/
50     return val;
51 }
```

3.7.4 tlb_write function

```
1     int tlbwrite(struct pcb_t *proc, BYTE data,
2         uint32_t destination, uint32_t offset)
3     {
4         int val;
5         // BYTE frmnum = -1;
6
7         int pgnum = PAGING_PGN(proc->mm->symrgtbl[destination].rg_start);
8         uint16_t frnum;
9         int return_value = tlb_cache_read(proc->tlb, proc->pid, pgnum, &frnum);
10        proc->tlb->total_access++;
11        /* TODO retrieve TLB CACHED frame num of accessing page(s)*/
12        /* by using tlb_cache_read()/tlb_cache_write()
13        frmnum is return value of tlb_cache_read/write value*/
14    #ifndef IODUMP
15        if (return_value >= 0)
16            printf("TLB hit at write region=%d offset=%d value=%d\n",
17                destination, offset, data);
18        else
19            printf("TLB miss at write region=%d offset=%d value=%d\n",
20                destination, offset, data);
21    #endif
22    #ifdef PAGETBL_DUMP
23        print_pgtbl(proc, 0, -1); // print max TBL
24    #endif
25    #ifdef MEMPHY_dump
26        MEMPHY_dump(proc->mram);
27    #endif
28    {
29        if (return_value >= 0)
30        {
31            proc->tlb->hit++;
32            int phyaddr = (frnum << PAGING_ADDR_FPN_LOBIT) + offset;
33            val = MEMPHY_write(proc->mram, phyaddr, data);
34        }
35        else
36        {
37            //__write
38            val = __write(proc, 0, destination, offset, data);
39            tlb_cache_write(proc->tlb, proc->pid, pgnum, PAGING_FPN_v2(proc->mm->pgd[
40                pgnum]));
41        }
42    }
43    /* TODO update TLB CACHED with frame num of recent accessing page(s)*/
44    /* by using tlb_cache_read()/tlb_cache_write()*/
45    return val;
46 }
```



45

3.8 Execution Result

3.8.1 Testcase os_0_mlq_paging

Input:

```
1 6 1 2
2 1048576 16777216 0 0 0
3 0 p0s 0
4 2 p1s 15
5
```

Output:

```
1 Time slot 0
2 ld_routine
3     Loaded a process at input/proc/p0s, PID: 1 PRI0: 0
4 Time slot 1
5     CPU 0: Dispatched process 1
6 ==CALC==
7 Time slot 2
8 ==ALLOC==
9     Loaded a process at input/proc/p1s, PID: 2 PRI0: 15
10 alloc 2 page
11 write TLB pgnum=0, fpn_num=1, pid=1
12
13 =print TLB=
14 88000001 ->pgnum:0, fpn_num:1, pid:1
15 write TLB pgnum=1, fpn_num=0, pid=1
16
17 =print TLB=
18 88000001 ->pgnum:0, fpn_num:1, pid:1
19 88002000 ->pgnum:1, fpn_num:0, pid:1
20 Time slot 3
21 ==ALLOC==
22 alloc 2 page
23 write TLB pgnum=2, fpn_num=3, pid=1
24
25 =print TLB=
26 88000001 ->pgnum:0, fpn_num:1, pid:1
27 88002000 ->pgnum:1, fpn_num:0, pid:1
28 88004003 ->pgnum:2, fpn_num:3, pid:1
29 write TLB pgnum=3, fpn_num=2, pid=1
30
31 =print TLB=
32 88000001 ->pgnum:0, fpn_num:1, pid:1
33 88002000 ->pgnum:1, fpn_num:0, pid:1
34 88004003 ->pgnum:2, fpn_num:3, pid:1
35 88006002 ->pgnum:3, fpn_num:2, pid:1
36 Time slot 4
37 ==FREE==
38 free TLB pgnum=0, fpn_num=1,pid=1
39
40 =print TLB=
41 88002000 ->pgnum:1, fpn_num:0, pid:1
42 88004003 ->pgnum:2, fpn_num:3, pid:1
43 88006002 ->pgnum:3, fpn_num:2, pid:1
44 free TLB pgnum=1, fpn_num=0,pid=1
45
```



```
46 =print TLB=
47 88004003 ->pgnum:2, fpn_num:3, pid:1
48 88006002 ->pgnum:3, fpn_num:2, pid:1
49 Time slot 5
50 ==ALLOC==
51 alloc 1 page
52 write TLB pgnum=1, fpn_num=0, pid=1
53
54 =print TLB=
55 88002000 ->pgnum:1, fpn_num:0, pid:1
56 88004003 ->pgnum:2, fpn_num:3, pid:1
57 88006002 ->pgnum:3, fpn_num:2, pid:1
58 Time slot 6
59 ==WRITE==
60 write data pgn=1, fpn_num=0, pid=1, data=100, phyaddr=20
61 TLB hit at write region=1 offset=20 value=100
62
63 =print_pgtbl: 0 - 1024=
64 00000000: 80000001
65 00000004: 80000000
66 00000008: 80000003
67 00000012: 80000002
68
69 =physical memdump=
70 BYTE 00000014: 100
71 Time slot 7
72 CPU 0: Put process 1 to run queue
73 CPU 0: Dispatched process 1
74 ==READ==
75 TLB hit at read region=1 offset=20
76
77 =print_pgtbl: 0 - 1024=
78 00000000: 80000001
79 00000004: 80000000
80 00000008: 80000003
81 00000012: 80000002
82
83 =physical memdump=
84 BYTE 00000014: 100
85 read data pgn=1, fpn_num=0, pid=1, data=100, phyaddr=20
86 Time slot 8
87 ==WRITE==
88 write TLB pgnum=0, fpn_num=1, pid=1
89
90 =print TLB=
91 88000001 ->pgnum:0, fpn_num:1, pid:1
92 88002000 ->pgnum:1, fpn_num:0, pid:1
93 88004003 ->pgnum:2, fpn_num:3, pid:1
94 88006002 ->pgnum:3, fpn_num:2, pid:1
95 write data pgn=0, fpn_num=1, pid=1, data=103
96 TLB miss at write region=3 offset=20 value=103
97
98 =print_pgtbl: 0 - 1024=
99 00000000: 80000001
100 00000004: 80000000
101 00000008: 80000003
102 00000012: 80000002
103
104 =physical memdump=
105 BYTE 00000014: 100
106 BYTE 00000114: 103
107 Time slot 9
```



```
108 ==READ==
109 TLB hit at read region=3 offset=20
110
111 =print_pgtbl: 0 - 1024=
112 00000000: 80000001
113 00000004: 80000000
114 00000008: 80000003
115 00000012: 80000002
116
117 =physical memdump=
118 BYTE 00000014: 100
119 BYTE 00000114: 103
120 read data pgn=0, fpn_num=1,pid=1, data=103, phyaddr=276
121 Time slot 10
122 ==FREE==
123 free TLB pgnum=2, fpn_num=3,pid=1
124
125 =print TLB=
126 88000001 ->pgnum:0, fpn_num:1, pid:1
127 88002000 ->pgnum:1, fpn_num:0, pid:1
128 88006002 ->pgnum:3, fpn_num:2, pid:1
129 free TLB pgnum=3, fpn_num=2,pid=1
130
131 =print TLB=
132 88000001 ->pgnum:0, fpn_num:1, pid:1
133 88002000 ->pgnum:1, fpn_num:0, pid:1
134 Time slot 11
135     CPU 0: Processed 1 has finished
136     CPU 0: Dispatched process 2
137 ==CALC==
138 Time slot 12
139 ==CALC==
140 Time slot 13
141 ==CALC==
142 Time slot 14
143 ==CALC==
144 Time slot 15
145 ==CALC==
146 Time slot 16
147 ==CALC==
148 Time slot 17
149     CPU 0: Put process 2 to run queue
150     CPU 0: Dispatched process 2
151 ==CALC==
152 Time slot 18
153 ==CALC==
154 Time slot 19
155 ==CALC==
156 Time slot 20
157 ==CALC==
158 Time slot 21
159     CPU 0: Processed 2 has finished
160     CPU 0 stopped
161
162 Numbers of TLB HIT: 3
163 Numbers of TLB MISS: 1
164 Numbers of TLB access: 4
165 TLB HIT rate: 0.750000
166 TLB MISS rate: 0.250000
167
168 =print TLB=
169 88000001 ->pgnum:0, fpn_num:1, pid:1
```



```
170 88002000 ->pgnum:1, fpn_num:0, pid:1
171
```

3.8.2 Testcase os_1_mlq_paging_small_1K

Input:

```
1 2 4 8
2 2048 16777216 0 0 0
3 1 p0s 130
4 2 s3 39
5 4 m1s 15
6 6 s2 120
7 7 m0s 120
8 9 p1s 15
9 11 s0 38
10 16 s1 0
11
```

Output:

```
1 Time slot 0
2 ld_routine
3 Time slot 1
4     Loaded a process at input/proc/p0s, PID: 1 PRI0: 130
5     CPU 3: Dispatched process 1
6 ==CALC==
7 Time slot 2
8     Loaded a process at input/proc/s3, PID: 2 PRI0: 39
9 ==ALLOC==
10 Time slot 3
11     CPU 2: Dispatched process 2
12 alloc 2 page
13 write TLB pgnum=0, fpn_num=1, pid=1
14
15 =print TLB=
16 88000001 ->pgnum:0, fpn_num:1, pid:1
17 ==CALC==
18 write TLB pgnum=1, fpn_num=0, pid=1
19
20 =print TLB=
21 88000001 ->pgnum:0, fpn_num:1, pid:1
22 88002000 ->pgnum:1, fpn_num:0, pid:1
23 ==CALC==
24     CPU 3: Put process 1 to run queue
25     CPU 3: Dispatched process 1
26 ==ALLOC==
27 alloc 2 page
28 write TLB pgnum=2, fpn_num=3, pid=1
29
30 =print TLB=
31 88000001 ->pgnum:0, fpn_num:1, pid:1
32 88002000 ->pgnum:1, fpn_num:0, pid:1
33 88004003 ->pgnum:2, fpn_num:3, pid:1
34 Time slot 4
35 write TLB pgnum=3, fpn_num=2, pid=1
36
37 =print TLB=
38 88000001 ->pgnum:0, fpn_num:1, pid:1
39 88002000 ->pgnum:1, fpn_num:0, pid:1
40 88004003 ->pgnum:2, fpn_num:3, pid:1
```



```
41 88006002 ->pgnum:3, fpn_num:2, pid:1
42     Loaded a process at input/proc/mis, PID: 3 PRI0: 15
43     CPU 2: Put process 2 to run queue
44 ==FREE==
45 free TLB pgnum=0, fpn_num=1,pid=1
46     CPU 2: Dispatched process 2
47 ==CALC==
48 Time slot 5
49     CPU 1: Dispatched process 3
50 ==ALLOC==
51
52 =print TLB=
53 88002000 ->pgnum:1, fpn_num:0, pid:1
54 88004003 ->pgnum:2, fpn_num:3, pid:1
55 88006002 ->pgnum:3, fpn_num:2, pid:1
56 alloc 2 page
57 write TLB pgnum=0, fpn_num=5, pid=3
58
59 =print TLB=
60 98000005 ->pgnum:0, fpn_num:5, pid:3
61 88002000 ->pgnum:1, fpn_num:0, pid:1
62 88004003 ->pgnum:2, fpn_num:3, pid:1
63 88006002 ->pgnum:3, fpn_num:2, pid:1
64 free TLB pgnum=1, fpn_num=0,pid=1
65
66 =print TLB=
67 98000005 ->pgnum:0, fpn_num:5, pid:3
68 88004003 ->pgnum:2, fpn_num:3, pid:1
69 88006002 ->pgnum:3, fpn_num:2, pid:1
70 write TLB pgnum=1, fpn_num=4, pid=3
71
72 =print TLB=
73 98000005 ->pgnum:0, fpn_num:5, pid:3
74 98002004 ->pgnum:1, fpn_num:4, pid:3
75 88004003 ->pgnum:2, fpn_num:3, pid:1
76 88006002 ->pgnum:3, fpn_num:2, pid:1
77     CPU 3: Put process 1 to run queue
78     Loaded a process at input/proc/s2, PID: 4 PRI0: 120
79 ==ALLOC==
80 alloc 1 page
81 Time slot 6
82 write TLB pgnum=1, fpn_num=4, pid=3
83
84 =print TLB=
85 98000005 ->pgnum:0, fpn_num:5, pid:3
86 98002004 ->pgnum:1, fpn_num:4, pid:3
87 88004003 ->pgnum:2, fpn_num:3, pid:1
88 88006002 -> CPU 3: Dispatched process 1
89 ==ALLOC==
90 alloc 1 page
91 pgnum:3, fpn_num:2, pid:1
92 ==CALC==
93 write TLB pgnum=1, fpn_num=0, pid=1
94
95 =print TLB=
96 98000005 ->pgnum:0, fpn_num:5, pid:3
97 88002000 ->pgnum:1, fpn_num:0, pid:1
98 88004003 ->pgnum:2, fpn_num:3, pid:1
99 88006002 ->pgnum:3, fpn_num:2, pid:1
100     CPU 1: Put process 3 to run queue
101     CPU 1: Dispatched process 3
102     CPU 2: Put process 2 to run queue
```




```
103 CPU 2: Dispatched process 2
104 ==CALC==
105 CPU 0: Dispatched process 4
106 ==CALC==
107 ==FREE==
108 ==WRITE==
109 write data pgn=1, fpn_num=0, pid=1, data=100, phyaddr=20
110 TLB hit at write region=1 offset=20 value=100
111
112 =print_pgtbl: 0 - 1024=
113 00000000: 80000001
114 00000004: 80000000
115 00000008: 80000003
116 00000012: 80000002
117
118 =physical memdump=
119 BYTE 00000014: 100
120 free TLB pgnum=0, fpn_num=5,pid=3
121
122 =print TLB=
123 Time slot 7
124 88002000 ->pgnum:1, fpn_num:0, pid:1
125 88004003 ->pgnum:2, fpn_num:3, pid:1
126 88006002 ->pgnum:3, fpn_num:2, pid:1
127 free TLB pgnum=1, fpn_num=0,pid=1
128
129 =print TLB=
130 88004003 ->pgnum:2, fpn_num:3, pid:1
131 88006002 ->pgnum:3, fpn_num:2, pid:1
132 Loaded a process at input/proc/m0s, PID: 5 PRI0: 120
133 CPU 3: Put process 1 to run queue
134 ==ALLOC==
135 alloc 1 page
136 ==CALC==
137 ==CALC==
138 Time slot 8
139 write TLB pgnum=0, fpn_num=5, pid=3
140
141 =print TLB=
142 98000005 ->pgnum:0, fpn_num:5, pid:3
143 88004003 ->pgnum:2, fpn_num:3, pid:1
144 88006002 ->pgnum:3, fpn_num:2, pid:1
145 CPU 3: Dispatched process 5
146 ==ALLOC==
147 alloc 2 page
148 write TLB pgnum=0, fpn_num=7, pid=5
149
150 =print TLB=
151 a8000007 ->pgnum:0, fpn_num:7, pid:5
152 88004003 ->pgnum:2, fpn_num:3, pid:1
153 88006002 ->pgnum:3, fpn_num:2, pid:1
154 write TLB pgnum=1, fpn_num=6, pid=5
155
156 =print TLB=
157 a8000007 ->pgnum:0, fpn_num:7, pid:5
158 a8002006 ->pgnum:1, fpn_num:6, pid:5
159 88004003 ->pgnum:2, fpn_num:3, pid:1
160 88006002 ->pgnum:3, fpn_num:2, pid:1
161 Loaded a process at input/proc/p1s, PID: 6 PRI0: 15
162 ==ALLOC==
163 alloc 1 page
164 write TLB pgnum=1, fpn_num=6, pid=5
```



```
165
166 =print TLB=
167 a8000007 ->pgnum:0, fpn_num:7, pid:5
168     CPU 0: Put process 4 to run queue
169     CPU 2: Put process 2 to run queue
170     CPU 0: Dispatched process 6
171 Time slot 9
172 ==CALC==
173     CPU 1: Put process 3 to run queue
174     CPU 2: Dispatched process 2
175 ==CALC==
176 a8002006 ->pgnum:1, fpn_num:6, pid:5
177 88004003 ->pgnum:2, fpn_num:3, pid:1
178 88006002 ->pgnum:3, fpn_num:2, pid:1
179     CPU 1: Dispatched process 3
180 ==FREE==
181 free TLB pgnum=0, fpn_num=7,pid=5
182
183 =print TLB=
184 a8002006 ->pgnum:1, fpn_num:6, pid:5
185 88004003 ->pgnum:2, fpn_num:3, pid:1
186 88006002 ->pgnum:3, fpn_num:2, pid:1
187     CPU 3: Put process 5 to run queue
188     CPU 3: Dispatched process 4
189 ==CALC==
190 ==FREE==
191 free TLB pgnum=1, fpn_num=6,pid=5
192
193 =print TLB=
194 88004003 ->pgnum:2, fpn_num:3, pid:1
195 88006002 ->pgnum:3, fpn_num:2, pid:1
196 ==CALC==
197 Time slot 10
198 ==CALC==
199 ==CALC==
200     CPU 1: Put process 3 to run queue
201     CPU 0: Put process 6 to run queue
202     CPU 0: Dispatched process 6
203 ==CALC==
204     CPU 1: Dispatched process 3
205 Time slot 11
206 ==FREE==
207     Loaded a process at input/proc/s0, PID: 7 PRI0: 38
208     CPU 2: Put process 2 to run queue
209     CPU 2: Dispatched process 7
210 ==CALC==
211     CPU 3: Put process 4 to run queue
212 ==FREE==
213 Time slot 12
214 ==CALC==
215 ==CALC==
216     CPU 3: Dispatched process 2
217 ==CALC==
218 ==CALC==
219     CPU 1: Processed 3 has finished
220     CPU 1: Dispatched process 4
221 ==CALC==
222     CPU 2: Put process 7 to run queue
223     CPU 2: Dispatched process 7
224 ==CALC==
225 Time slot 13
226     CPU 0: Put process 6 to run queue
```



```
227 CPU 0: Dispatched process 6
228 ==CALC==
229 CPU 3: Put process 2 to run queue
230 Time slot 14
231 ==CALC==
232 ==CALC==
233 CPU 3: Dispatched process 2
234 ==CALC==
235 ==CALC==
236 CPU 3: Processed 2 has finished
237 Time slot 15
238 CPU 0: Put process 6 to run queue
239 CPU 0: Dispatched process 6
240 ==CALC==
241 CPU 3: Dispatched process 5
242 ==FREE==
243 CPU 2: Put process 7 to run queue
244 CPU 2: Dispatched process 7
245 ==CALC==
246 CPU 1: Put process 4 to run queue
247 CPU 1: Dispatched process 4
248 ==CALC==
249 ==ALLOC==
250 Time slot 16
251 alloc 1 page
252 write TLB pgnum=0, fpn_num=7, pid=5
253 Loaded a process at input/proc/s1, PID: 8 PRI0: 0
254 ==CALC==
255 ==CALC==
256
257 =print TLB=
258 a8000007 ->pgnum:0, fpn_num:7, pid:5
259 88004003 ->pgnum:2, fpn_num:3, pid:1
260 88006002 ->pgnum:3, fpn_num:2, pid:1
261 ==CALC==
262 CPU 3: Put process 5 to run queue
263 Time slot 17
264 CPU 0: Put process 6 to run queue
265 CPU 0: Dispatched process 8
266 ==ALLOC==
267 CPU 2: Put process 7 to run queue
268 CPU 2: Dispatched process 7
269 ==CALC==
270 CPU 3: Dispatched process 6
271 ==CALC==
272 CPU 1: Put process 4 to run queue
273 CPU 1: Dispatched process 4
274 ==CALC==
275 alloc 2 page
276 write TLB pgnum=0, fpn_num=0, pid=8
277
278 =print TLB=
279 c0000000 ->pgnum:0, fpn_num:0, pid:8
280 88004003 ->pgnum:2, fpn_num:3, pid:1
281 88006002 ->pgnum:3, fpn_num:2, pid:1
282 write TLB pgnum=1, fpn_num=0, pid=8
283
284 =print TLB=
285 c0000000 ->pgnum:0, fpn_num:0, pid:8
286 c0002000 ->pgnum:1, fpn_num:0, pid:8
287 88004003 ->pgnum:2, fpn_num:3, pid:1
288 88006002 ->pgnum:3, fpn_num:2, pid:1
```



```
289 ==CALC==
290 ==CALC==
291 ==CALC==
292 ==CALC==
293 Time slot 18
294     CPU 3: Processed 6 has finished
295     CPU 3: Dispatched process 5
296     CPU 2: Put process 7 to run queue
297 Time slot 19
298     CPU 1: Put process 4 to run queue
299     CPU 1: Dispatched process 4
300 ==CALC==
301 ==WRITE==
302 write TLB pgnum=1, fpn_num=6, pid=5
303
304 =print TLB=
305 c0000000 ->pgnum:0, fpn_num:0, pid:8
306 a8002006 ->pgnum:1, fpn_num:6, pid:5
307 88004003 ->pgnum:2, fpn_num:3, pid:1
308 88006002 ->pgnum:3, fpn_num:2, pid:1
309     CPU 0: Put process 8 to run queue
310 write data pgn=1, fpn_num=6, pid=5, data=102
311 TLB miss at write region=1 offset=20 value=102
312
313 =print_pgtbl: 0 - 512=
314 00000000: 80000007
315 00000004: 80000006
316
317 =physical memdump=
318 BYTE 00000014: 100
319     CPU 0: Dispatched process 8
320     CPU 2: Dispatched process 7
321 ==CALC==
322 BYTE 000006a4: 102
323 ==CALC==
324 ==WRITE==
325 ==CALC==
326 Time slot 20
327 ==CALC==
328 write TLB pgnum=0, fpn_num=0, pid=5
329
330 =print TLB=
331 a8000000 ->pgnum:0, fpn_num:0, pid:5
332 a8002006 ->pgnum:1, fpn_num:6, pid:5
333 88004003 ->pgnum:2, fpn_num:3, pid:1
334 88006002 ->pgnum:3, fpn_num:2, pid:1
335 ==CALC==
336 write data pgn=0, fpn_num=0, pid=5, data=1
337 TLB miss at write region=2 offset=1000 value=1
338
339 =print_pgtbl: 0 - 512=
340 00000000: c0000000
341 00000004: 80000006
342
343 =physical memdump=
344 BYTE 00000014: 100
345 BYTE 000000b0: 1
346 BYTE 000006a4: 102
347     CPU 3: Put process 5 to run queue
348     CPU 1: Processed 4 has finished
349     CPU 0: Put process 8 to run queue
350     CPU 2: Put process 7 to run queue
```



```
351 Time slot 21
352     CPU 0: Dispatched process 8
353     CPU 1: Dispatched process 1
354 ==READ==
355 TLB miss at read region=1 offset=20
356
357 =print_pgtbl: 0 - 1024=
358 00000000: 80000001
359 00000004: 80000000
360 00000008: 80000003
361 00000012: 80000002
362
363 =physical memdump=
364 BYTE 00000014: 100
365 BYTE 000000b0: 1
366 BYTE 000006a4: 102
367 write TLB pgnum=1, fpn_num=0, pid=1
368
369 =print TLB=
370 a8000000 ->pgnum:0, fpn_num:0, pid:5
371 88002000 ->pgnum:1, fpn_num:0, pid:1
372 88004003 ->pgnum:2, fpn_num:3, pid:1
373 88006002 ->pgnum:3, fpn_num:2, pid:1
374 ==CALC==
375 read data pgn=1, fpn_num=0,pid=1, data=0
376     CPU 3: Dispatched process 5
377 ==WRITE==
378 write data pgn=0, fpn_num=0, pid=5, data=0, phyaddr=0
379 TLB hit at write region=0 offset=0 value=0
380
381     CPU 2: Dispatched process 7
382 ==CALC==
383 =print_pgtbl: 0 - 512=
384 00000000: c0000000
385 00000004: 80000006
386
387 =physical memdump=
388 BYTE 00000014: 100
389 BYTE 000000b0: 1
390 BYTE 000006a4: 102
391     CPU 3: Processed 5 has finished
392 ==CALC==
393 ==CALC==
394 Time slot 22
395     CPU 3 stopped
396 ==WRITE==
397 write TLB pgnum=0, fpn_num=1, pid=1
398
399 =print TLB=
400 88000001 ->pgnum:0, fpn_num:1, pid:1
401 88002000 ->pgnum:1, fpn_num:0, pid:1
402 88004003 ->pgnum:2, fpn_num:3, pid:1
403 88006002 ->pgnum:3, fpn_num:2, pid:1
404 write data pgn=0, fpn_num=1, pid=1, data=103
405 TLB miss at write region=3 offset=20 value=103
406
407 =print_pgtbl: 0 - 1024=
408 00000000: 80000001
409 00000004: 80000000
410 00000008: 80000003
411 00000012: 80000002
412
```



```
413 =physical memdump=
414 BYTE 00000014: 100
415 BYTE 000000b0: 1
416 BYTE 00000114: 103
417 BYTE 000006a4: 102
418     CPU 1: Put process 1 to run queue
419     CPU 1: Dispatched process 1
420 ==READ==
421 TLB hit at read region=3 offset=20
422
423 =print_pgtbl: 0 - 1024=
424 00000000: 80000001
425 00000004: 80000000
426 00000008: 80000003
427 00000012: 80000002
428
429 =physical memdump=
430 BYTE 00000014: 100
431 BYTE 000000b0: 1
432 BYTE 00000114: 103
433 BYTE 000006a4: 102
434 read data pgn=0, fpn_num=1,pid=1, data=103, phyaddr=276
435 Time slot 23
436     CPU 0: Put process 8 to run queue
437     CPU 0: Dispatched process 8
438 ==CALC==
439     CPU 2: Put process 7 to run queue
440     CPU 2: Dispatched process 7
441 ==CALC==
442 ==CALC==
443 Time slot 24
444     CPU 0: Processed 8 has finished
445     CPU 0 stopped
446 ==FREE==
447 free TLB pgnum=2, fpn_num=3,pid=1
448
449 =print TLB=
450 88000001 ->pgnum:0, fpn_num:1, pid:1
451 88002000 ->pgnum:1, fpn_num:0, pid:1
452 88006002 ->pgnum:3, fpn_num:2, pid:1
453 free TLB pgnum=3, fpn_num=2,pid=1
454
455 =print TLB=
456 88000001 ->pgnum:0, fpn_num:1, pid:1
457 88002000 ->pgnum:1, fpn_num:0, pid:1
458     CPU 2: Put process 7 to run queue
459     CPU 2: Dispatched process 7
460 ==CALC==
461 Time slot 25
462     CPU 1: Processed 1 has finished
463     CPU 1 stopped
464 Time slot 26
465     CPU 2: Processed 7 has finished
466     CPU 2 stopped
467
468 Numbers of TLB HIT: 3
469 Numbers of TLB MISS: 4
470 Numbers of TLB access: 7
471 TLB HIT rate: 0.428571
472 TLB MISS rate: 0.571429
473
474 =print TLB=
```



```
475 88000001 ->pgnum:0, fpn_num:1, pid:1
476 88002000 ->pgnum:1, fpn_num:0, pid:1
477
```

4 Put it all together

4.1 Synchronization

4.1.1 Design

Concept: Ensuring parallel processes do not interfere with each other. This involves:

- Mutual exclusion: Only one process can access a non-shareable resource at any given time (critical section).
- Synchronization: Processes need to cooperate to accomplish tasks.

A section of code where errors can occur when accessing shared resources (variables, files, etc.) is called a critical region.

When solving critical region problems, consider these four conditions:

- No two processes can be inside the critical region simultaneously.
- No assumptions about the speed of processes or the number of processors.
- A process outside the critical region should not prevent other processes from entering.
- No process should have to wait indefinitely to enter the critical region.

A Mutex lock (or simply mutex) is a synchronization mechanism used to ensure that only one process can access a specific resource at a time. In order to implement process synchronization, this project utilizes a mutex lock to protect resources during program execution.

4.1.2 Implementation

Finally, we combine Scheduler and Memory Management to form a complete OS. . The last task to do is synchronization. Since the OS runs on multiple processors, it is possible that share resources could be concurrently accessed by more than one process at a time. Your job in this section is to find share resource and use lock mechanism to protect them. Check your work by fist compiling the whole source code and compare your output with those in output. Remember that as we are running multiple processes, there may be more than one correct result.

We will insert lock mechanism in: `tlb_cache_read()`, `tlb_cache_write()`, `__read()`, `__alloc()`, `__free()`, `free_pcb_memph()`, `get_mlq_proc()`, `put_mlq_proc()`, `add_mlq_proc()`, `get_proc()`, `put_proc()`, `add_proc()`.

Testcase os_1_mlq_paging_small_1K Input:

```
1 2 4 8
2 2048 16777216 0 0 0
3 1 p0s 130
4 2 s3 39
5 4 m1s 15
6 6 s2 120
7 7 m0s 120
8 9 p1s 15
```



```
9 11 s0 38
10 16 s1 0
11
```

4.1.2.a Without synchronization:

```
1 Time slot 0
2 ld_routine
3     Loaded a process at input/proc/p0s, PID: 1 PRI0: 130
4 Time slot 1
5     CPU 3: Dispatched process 1
6 ==CALC==
7 Time slot 2
8     Loaded a process at input/proc/s3, PID: 2 PRI0: 39
9 ==ALLOC==
10    CPU 2: Dispatched process 2
11 alloc 2 page
12 write TLB pgnum=0, fpn_num=1, pid=1
13
14 =print TLB=
15 88000001 ->pgnum:0, fpn_num:1, pid:1
16 ==CALC==
17 Time slot 3
18 write TLB pgnum=1, fpn_num=0, pid=1
19
20 =print TLB=
21 88000001 ->pgnum:0, fpn_num:1, pid:1
22 88002000 ->pgnum:1, fpn_num:0, pid:1
23     Loaded a process at input/proc/m1s, PID: 3 PRI0: 15
24 Time slot 4
25 ==CALC==
26     CPU 1: Dispatched process 3
27 ==ALLOC==
28     CPU 3: Put process 1 to run queue
29 alloc 2 page
30     CPU 3: Dispatched process 1
31 ==ALLOC==
32 write TLB pgnum=0, fpn_num=3, pid=3
33
34 =print TLB=
35 98000003 ->pgnum:0, fpn_num:3, pid:3
36 88002000 ->pgnum:1, fpn_num:0, pid:1
37 alloc 2 page
38 write TLB pgnum=2, fpn_num=5, pid=1
39
40 =print TLB=
41 98000003 ->pgnum:0, fpn_num:3, pid:3
42 88002000 ->pgnum:1, fpn_num:0, pid:1
43 88004005 ->pgnum:2, fpn_num:5, pid:1
44 write TLB pgnum=1, fpn_num=2, pid=3
45
46 =print TLB=
47 98000003 ->pgnum:0, fpn_num:3, pid:3
48 98002002 ->pgnum:1, fpn_num:2, pid:3
49 88004005 ->pgnum:2, fpn_num:5, pid:1
50 write TLB pgnum=3, fpn_num=4, pid=1
51
52 =print TLB=
53 98000003 ->pgnum:0, fpn_num:3, pid:3
54 98002002 ->pgnum:1, fpn_num:2, pid:3
55 88004005 ->pgnum:2, fpn_num:5, pid:1
```




```
56 88006004 ->pgnum:3, fpn_num:4, pid:1
57 ==FREE==
58 CPU 2: Put process 2 to run queue
59 Time slot 5
60 ==ALLOC==
61 alloc 1 page
62 write TLB pgnum=1, fpn_num=2, pid=3
63
64 =print TLB=
65 98000003 -> CPU 2: Dispatched process 2
66 ==CALC==
67 pgnum:0, fpn_num:3, pid:3
68 98002002 ->pgnum:1, fpn_num:2, pid:3
69 88004005 ->pgnum:2, fpn_num:5, pid:1
70 88006004 ->pgnum:3, fpn_num:4, pid:1
71 free TLB pgnum=0, fpn_num=3,pid=3
72
73 =print TLB=
74 98002002 ->pgnum:1, fpn_num:2, pid:3
75 88004005 ->pgnum:2, fpn_num:5, pid:1
76 88006004 ->pgnum:3, fpn_num:4, pid:1
77 free TLB pgnum=1, fpn_num=2,pid=3
78
79 =print TLB=
80 88004005 ->pgnum:2, fpn_num:5, pid:1
81 88006004 ->pgnum:3, fpn_num:4, pid:1
82 CPU 3: Put process 1 to run queue
83 Time slot 6
84 Loaded a process at input/proc/s2, PID: 4 PRI0: 120
85 CPU 1: Put process 3 to run queue
86 CPU 1: Dispatched process 3
87 ==FREE==
88 ==CALC==
89 CPU 3: Dispatched process 1
90 ==ALLOC==
91 alloc 1 page
92 write TLB pgnum=1, fpn_num=0, pid=1
93
94 =print TLB=
95 88002000 ->pgnum:1, fpn_num:0, pid:1
96 88004005 ->pgnum:2, fpn_num:5, pid:1
97 88006004 ->pgnum:3, fpn_num:4, pid:1
98 CPU 0: Dispatched process 1
99 ==WRITE==
100 write data pgn=1, fpn_num=0, pid=1, data=100, phyaddr=20
101 TLB hit at write region=1 offset=20 value=100
102
103 =print_pgtbl: 0 - 1024=
104 00000000: 80000001
105 00000004: 80000000
106 00000008: 80000005
107 00000012: 80000004
108
109 =physical memdump=
110 BYTE 00000014: 100
111 ==READ==
112 Time slot 7
113 ==ALLOC==
114 alloc 1 page
115 write TLB pgnum=0, fpn_num=3, pid=3
116
117 =print TLB=
```



```
118 98000003 ->pgnum:0, fpn_num:3, pid:3
119 88002000 ->pgnum:1, fpn_num:0, pid:1
120 88004005 ->pgnum:2, fpn_num:5, pid:1
121 88006004 ->pgnum:3, fpn_num:4, pid:1
122 ==WRITE==
123 write TLB pgnum=0, fpn_num=1, pid=1
124
125 =print TLB=
126 88000001 ->pgnum:0, fpn_num:1, pid:1
127 88002000 ->pgnum:1, fpn_num:0, pid:1
128 88004005 ->pgnum:2, fpn_num:5, pid:1
129 88006004 ->pgnum:3, fpn_num:4, pid:1
130 TLB hit at read region=1 offset=20
131
132 =print_pgtbl: 0 - 1024=
133 00000000: 80000001
134 00000004: 80000000
135 00000008: 80000005
136 00000012: 80000004
137
138 =physical memdump=
139 BYTE 00000014: 100
140 BYTE 00000114: 103
141 read data pgn=1, fpn_num=0,pid=1, data=100, phyaddr=20
142     CPU 2: Put process 2 to run queue
143     CPU 2: Dispatched process 2
144 ==CALC==
145 write data pgn=0, fpn_num=1, pid=1, data=103
146 TLB miss at write region=3 offset=20 value=103
147
148 =print_pgtbl: 0 - 1024=
149 00000000: 80000001
150 00000004: 80000000
151 00000008: 80000005
152 00000012: 80000004
153
154 =physical memdump=
155 BYTE 00000014: 100
156 BYTE 00000114: 103
157     Loaded a process at input/proc/m0s, PID: 5 PRI0: 120
158     CPU 3: Put process 1 to run queue
159     CPU 1: Put process 3 to run queue
160     CPU 1: Dispatched process 3
161 ==FREE==
162 ==CALC==
163     CPU 0: Put process 1 to run queue
164     CPU 0: Dispatched process 5
165 ==ALLOC==
166 free TLB pgnum=0, fpn_num=1,pid=1
167
168 =print TLB=
169 88002000 ->pgnum:1, fpn_num:0, pid:1
170 88004005 ->pgnum:2, fpn_num:5, pid:1
171 88006004 ->pgnum:3, fpn_num:4, pid:1
172 alloc 2 page
173 Time slot 8
174     CPU 3: Dispatched process 4
175 write TLB pgnum=0, fpn_num=7, pid=5
176
177 =print TLB=
178 a8000007 ->pgnum:0, fpn_num:7, pid:5
179 88002000 ->pgnum:1, fpn_num:0, pid:1
```



```
180 88004005 ->pgnum:2, fpn_num:5, pid:1
181 88006004 ->pgnum:3, fpn_num:4, pid:1
182 ==CALC==
183 write TLB pgnum=1, fpn_num=6, pid=5
184
185 =print TLB=
186 a8000007 ->pgnum:0, fpn_num:7, pid:5
187 a8002006 ->pgnum:1, fpn_num:6, pid:5
188 88004005 ->pgnum:2, fpn_num:5, pid:1
189 88006004 ->pgnum:3, fpn_num:4, pid:1
190     Loaded a process at input/proc/p1s, PID: 6 PRI0: 15
191 Time slot 9
192 ==ALLOC==
193 alloc 1 page
194     CPU 2: Put process 2 to run queue
195     CPU 2: Dispatched process 6
196 ==CALC==
197 write TLB pgnum=1, fpn_num=6, pid=5
198
199 =print TLB=
200 a8000007 ->pgnum:0, fpn_num:7, pid:5
201 a8002006 ->pgnum:1, fpn_num:6, pid:5
202 88004005 ->pgnum:2, fpn_num:5, pid:1
203 88006004 ->pgnum:3, fpn_num:4, pid:1
204 ==FREE==
205 free TLB pgnum=1, fpn_num=6, pid=5
206
207 =print TLB=
208 a8000007 ->pgnum:0, fpn_num:7, pid:5
209 88004005 ->pgnum:2, fpn_num:5, pid:1
210 88006004 ->pgnum:3, fpn_num:4, pid:1
211 ==CALC==
212     CPU 3: Put process 4 to run queue
213     CPU 3: Dispatched process 2
214 ==CALC==
215     CPU 1: Put process 3 to run queue
216 Time slot 10
217     CPU 0: Put process 5 to run queue
218     CPU 0: Dispatched process 4
219 ==CALC==
220     CPU 1: Dispatched process 3
221 ==FREE==
222 ==CALC==
223     Loaded a process at input/proc/s0, PID: 7 PRI0: 38
224 Time slot 11
225     CPU 2: Put process 6 to run queue
226     CPU 2: Dispatched process 6
227 ==CALC==
228 ==CALC==
229 ==FREE==
230 ==CALC==
231 ==CALC==
232     CPU 3: Put process 2 to run queue
233     CPU 3: Dispatched process 7
234 ==CALC==
235     CPU 0: Put process 4 to run queue
236     CPU 0: Dispatched process 2
237 ==CALC==
238     CPU 1: Processed 3 has finished
239     CPU 1: Dispatched process 4
240 ==CALC==
241 Time slot 12
```



```
242 ==CALC==
243 ==CALC==
244 ==CALC==
245 Time slot 13
246     CPU 2: Put process 6 to run queue
247     CPU 2: Dispatched process 6
248 ==CALC==
249 ==CALC==
250     CPU 1: Put process 4 to run queue
251     CPU 3: Put process 7 to run queue
252     CPU 3: Dispatched process 7
253 ==CALC==
254     CPU 1: Dispatched process 4
255 ==CALC==
256     CPU 0: Put process 2 to run queue
257     CPU 0: Dispatched process 2
258 ==CALC==
259 Time slot 14
260 ==CALC==
261 ==CALC==
262 Time slot 15
263     CPU 2: Put process 6 to run queue
264     CPU 2: Dispatched process 6
265 ==CALC==
266     CPU 0: Processed 2 has finished
267     CPU 0: Dispatched process 5
268 ==FREE==
269 free TLB pgnum=0, fpn_num=7,pid=5
270
271 =print TLB=
272 88004005 ->pgnum:2, fpn_num:5, pid:1
273 88006004 ->pgnum:3, fpn_num:4, pid:1
274     CPU 3: Put process 7 to run queue
275     CPU 3: Dispatched process 7
276 ==CALC==
277 Time slot 16
278     CPU 1: Put process 4 to run queue
279     CPU 1: Dispatched process 4
280 ==CALC==
281 ==ALLOC==
282 alloc 1 page
283 write TLB pgnum=0, fpn_num=7, pid=5
284
285 =print TLB=
286 a8000007 ->pgnum:0, fpn_num:7, pid:5
287 88004005 ->pgnum:2, fpn_num:5, pid:1
288 88006004 ->pgnum:3, fpn_num:4, pid:1
289     Loaded a process at input/proc/s1, PID: 8 PRI0: 0
290 ==CALC==
291 ==CALC==
292 ==CALC==
293 Time slot 17
294     CPU 0: Put process 5 to run queue
295     CPU 0: Dispatched process 8
296 ==ALLOC==
297 alloc 2 page
298 write TLB pgnum=0, fpn_num=0, pid=8
299
300 =print TLB=
301 c0000000 ->pgnum:0, fpn_num:0, pid:8
302 88004005 ->pgnum:2, fpn_num:5, pid:1
303 88006004 ->pgnum:3, fpn_num:4, pid:1
```



```
304         CPU 2: Put process 6 to run queue
305         CPU 2: Dispatched process 6
306 write TLB pgnum=1, fpn_num=0, pid=8
307
308 =print TLB=
309 c0000000 ->pgnum:0, fpn_num:0, pid:8
310 c0002000 ->pgnum:1, fpn_num:0, pid:8
311 88004005 ->pgnum:2, fpn_num:5, pid:1
312 88006004 ->pgnum:3, fpn_num:4, pid:1
313 ==CALC==
314         CPU 3: Put process 7 to run queue
315 Time slot 18
316         CPU 1: Put process 4 to run queue
317         CPU 1: Dispatched process 4
318 ==CALC==
319 ==CALC==
320         CPU 3: Dispatched process 7
321 ==CALC==
322 ==CALC==
323 ==CALC==
324 ==CALC==
325 Time slot 19
326         CPU 0: Put process 8 to run queue
327         CPU 0: Dispatched process 8
328 ==CALC==
329         CPU 2: Processed 6 has finished
330         CPU 2: Dispatched process 5
331 ==WRITE==
332 write TLB pgnum=1, fpn_num=6, pid=5
333
334 =print TLB=
335 c0000000 ->pgnum:0, fpn_num:0, pid:8
336 a8002006 ->pgnum:1, fpn_num:6, pid:5
337 88004005 ->pgnum:2, fpn_num:5, pid:1
338 88006004 ->pgnum:3, fpn_num:4, pid:1
339 write data pgn=1, fpn_num=6, pid=5, data=102
340 TLB miss at write region=1 offset=20 value=102
341
342 =print_pgtbl: 0 - 512=
343 00000000: 80000007
344 00000004: 80000006
345
346 =physical memdump=
347 BYTE 00000014: 100
348 BYTE 00000114: 103
349 BYTE 000006a4: 102
350         CPU 3: Put process 7 to run queue
351         CPU 1: Processed 4 has finished
352 ==CALC==
353 Time slot 20
354 ==WRITE==
355 write TLB pgnum=0, fpn_num=0, pid=5
356
357 =print TLB=
358         CPU 1: Dispatched process 1
359 ==READ==
360 TLB miss at read region=3 offset=20
361
362 =print_pgtbl: 0 - 1024=
363 00000000: 80000001
364 00000004: 80000000
365 00000008: 80000005
```



```
366 00000012: 80000004
367
368 =physical memdump=
369 BYTE 00000014: 100
370 BYTE 000000b0: 1
371 BYTE 00000114: 103
372 BYTE 000006a4: 102
373 write TLB pgnum=0, fpn_num=1, pid=1
374
375 =print TLB=
376 88000001 ->pgnum:0, fpn_num:1, pid:1
377 a8002006 ->pgnum:1, fpn_num:6, pid:5
378 88004005 ->pgnum:2, fpn_num:5, pid:1
379 88006004 ->pgnum:3, fpn_num:4, pid:1
380 CPU 3: Dispatched process 7
381 ==CALC==
382 read data pgnum=0, fpn_num=1,pid=1, data=103
383 a8000000 ->pgnum:0, fpn_num:1, pid:1
384 a8002006 ->pgnum:1, fpn_num:6, pid:5
385 88004005 ->pgnum:2, fpn_num:5, pid:1
386 88006004 ->pgnum:3, fpn_num:4, pid:1
387 write data pgnum=0, fpn_num=1, pid=1, data=1
388 TLB miss at write region=2 offset=1000 value=1
389
390 =print_pgtbl: 0 - 512=
391 00000000: c0000000
392 00000004: 80000006
393
394 =physical memdump=
395 BYTE 00000014: 100
396 BYTE 000000b0: 1
397 BYTE 00000114: 103
398 BYTE 000006a4: 102
399 ==CALC==
400 ==FREE==
401 free TLB pgnum=2, fpn_num=5,pid=1
402
403 =print TLB=
404 88000001 ->pgnum:0, fpn_num:1, pid:1
405 a8002006 ->pgnum:1, fpn_num:6, pid:5
406 88006004 ->pgnum:3, fpn_num:4, pid:1
407 Time slot 21
408 free TLB pgnum=3, fpn_num=4,pid=1
409
410 =print TLB=
411 88000001 ->pgnum:0, fpn_num:1, pid:1
412 a8002006 ->pgnum:1, fpn_num:6, pid:5
413 CPU 0: Put process 8 to run queue
414 CPU 0: Dispatched process 8
415 ==CALC==
416 CPU 2: Put process 5 to run queue
417 CPU 2: Dispatched process 5
418 ==WRITE==
419 write TLB pgnum=0, fpn_num=0, pid=5
420
421 =print TLB=
422 a8000000 ->pgnum:0, fpn_num:0, pid:5
423 a8002006 ->pgnum:1, fpn_num:6, pid:5
424 write data pgnum=0, fpn_num=0, pid=5, data=0
425 TLB miss at write region=0 offset=0 value=0
426
427 =print_pgtbl: 0 - 512=
```



```
428 00000000: c0000000
429 00000004: 80000006
430
431 =physical memdump=
432 BYTE 00000014: 100
433 BYTE 000000b0: 1
434 BYTE 00000114: 103
435 BYTE 000006a4: 102
436     CPU 3: Put process 7 to run queue
437     CPU 1: Processed 1 has finished
438 Time slot 22
439     CPU 3: Dispatched process 7
440 ==CALC==
441     CPU 1: Dispatched process 1542034800
442 ==CALC==
443     CPU 2: Processed 5 has finished
444     CPU 2 stopped
445 Segmentation fault
```

4.1.2.b With synchronization:

```
1 Time slot 0
2 ld_routine
3 Time slot 1
4     Loaded a process at input/proc/p0s, PID: 1 PRI0: 130
5     CPU 3: Dispatched process 1
6 ==CALC==
7 Time slot 2
8     Loaded a process at input/proc/s3, PID: 2 PRI0: 39
9 ==ALLOC==
10 Time slot 3
11     CPU 2: Dispatched process 2
12 alloc 2 page
13 write TLB pgnum=0, fpn_num=1, pid=1
14
15 =print TLB=
16 88000001 ->pgnum:0, fpn_num:1, pid:1
17 ==CALC==
18 write TLB pgnum=1, fpn_num=0, pid=1
19
20 =print TLB=
21 88000001 ->pgnum:0, fpn_num:1, pid:1
22 88002000 ->pgnum:1, fpn_num:0, pid:1
23 ==CALC==
24     CPU 3: Put process 1 to run queue
25     CPU 3: Dispatched process 1
26 ==ALLOC==
27 alloc 2 page
28 write TLB pgnum=2, fpn_num=3, pid=1
29
30 =print TLB=
31 88000001 ->pgnum:0, fpn_num:1, pid:1
32 88002000 ->pgnum:1, fpn_num:0, pid:1
33 88004003 ->pgnum:2, fpn_num:3, pid:1
34 Time slot 4
35 write TLB pgnum=3, fpn_num=2, pid=1
36
37 =print TLB=
38 88000001 ->pgnum:0, fpn_num:1, pid:1
39 88002000 ->pgnum:1, fpn_num:0, pid:1
40 88004003 ->pgnum:2, fpn_num:3, pid:1
```



```
41 88006002 ->pgnum:3, fpn_num:2, pid:1
42     Loaded a process at input/proc/mis, PID: 3 PRI0: 15
43     CPU 2: Put process 2 to run queue
44 ==FREE==
45 free TLB pgnum=0, fpn_num=1,pid=1
46     CPU 2: Dispatched process 2
47 ==CALC==
48 Time slot 5
49     CPU 1: Dispatched process 3
50 ==ALLOC==
51
52 =print TLB=
53 88002000 ->pgnum:1, fpn_num:0, pid:1
54 88004003 ->pgnum:2, fpn_num:3, pid:1
55 88006002 ->pgnum:3, fpn_num:2, pid:1
56 alloc 2 page
57 write TLB pgnum=0, fpn_num=5, pid=3
58
59 =print TLB=
60 98000005 ->pgnum:0, fpn_num:5, pid:3
61 88002000 ->pgnum:1, fpn_num:0, pid:1
62 88004003 ->pgnum:2, fpn_num:3, pid:1
63 88006002 ->pgnum:3, fpn_num:2, pid:1
64 free TLB pgnum=1, fpn_num=0,pid=1
65
66 =print TLB=
67 98000005 ->pgnum:0, fpn_num:5, pid:3
68 88004003 ->pgnum:2, fpn_num:3, pid:1
69 88006002 ->pgnum:3, fpn_num:2, pid:1
70 write TLB pgnum=1, fpn_num=4, pid=3
71
72 =print TLB=
73 98000005 ->pgnum:0, fpn_num:5, pid:3
74 98002004 ->pgnum:1, fpn_num:4, pid:3
75 88004003 ->pgnum:2, fpn_num:3, pid:1
76 88006002 ->pgnum:3, fpn_num:2, pid:1
77     CPU 3: Put process 1 to run queue
78     Loaded a process at input/proc/s2, PID: 4 PRI0: 120
79 ==ALLOC==
80 alloc 1 page
81 Time slot 6
82 write TLB pgnum=1, fpn_num=4, pid=3
83
84 =print TLB=
85 98000005 ->pgnum:0, fpn_num:5, pid:3
86 98002004 ->pgnum:1, fpn_num:4, pid:3
87 88004003 ->pgnum:2, fpn_num:3, pid:1
88 88006002 -> CPU 3: Dispatched process 1
89 ==ALLOC==
90 alloc 1 page
91 pgnum:3, fpn_num:2, pid:1
92 ==CALC==
93 write TLB pgnum=1, fpn_num=0, pid=1
94
95 =print TLB=
96 98000005 ->pgnum:0, fpn_num:5, pid:3
97 88002000 ->pgnum:1, fpn_num:0, pid:1
98 88004003 ->pgnum:2, fpn_num:3, pid:1
99 88006002 ->pgnum:3, fpn_num:2, pid:1
100     CPU 1: Put process 3 to run queue
101     CPU 1: Dispatched process 3
102     CPU 2: Put process 2 to run queue
```




```
103 CPU 2: Dispatched process 2
104 ==CALC==
105 CPU 0: Dispatched process 4
106 ==CALC==
107 ==FREE==
108 ==WRITE==
109 write data pgn=1, fpn_num=0, pid=1, data=100, phyaddr=20
110 TLB hit at write region=1 offset=20 value=100
111
112 =print_pgtbl: 0 - 1024=
113 00000000: 80000001
114 00000004: 80000000
115 00000008: 80000003
116 00000012: 80000002
117
118 =physical memdump=
119 BYTE 00000014: 100
120 free TLB pgnum=0, fpn_num=5, pid=3
121
122 =print TLB=
123 Time slot 7
124 88002000 ->pgnum:1, fpn_num:0, pid:1
125 88004003 ->pgnum:2, fpn_num:3, pid:1
126 88006002 ->pgnum:3, fpn_num:2, pid:1
127 free TLB pgnum=1, fpn_num=0, pid=1
128
129 =print TLB=
130 88004003 ->pgnum:2, fpn_num:3, pid:1
131 88006002 ->pgnum:3, fpn_num:2, pid:1
132 Loaded a process at input/proc/m0s, PID: 5 PRI0: 120
133 CPU 3: Put process 1 to run queue
134 ==ALLOC==
135 alloc 1 page
136 ==CALC==
137 ==CALC==
138 Time slot 8
139 write TLB pgnum=0, fpn_num=5, pid=3
140
141 =print TLB=
142 98000005 ->pgnum:0, fpn_num:5, pid:3
143 88004003 ->pgnum:2, fpn_num:3, pid:1
144 88006002 ->pgnum:3, fpn_num:2, pid:1
145 CPU 3: Dispatched process 5
146 ==ALLOC==
147 alloc 2 page
148 write TLB pgnum=0, fpn_num=7, pid=5
149
150 =print TLB=
151 a8000007 ->pgnum:0, fpn_num:7, pid:5
152 88004003 ->pgnum:2, fpn_num:3, pid:1
153 88006002 ->pgnum:3, fpn_num:2, pid:1
154 write TLB pgnum=1, fpn_num=6, pid=5
155
156 =print TLB=
157 a8000007 ->pgnum:0, fpn_num:7, pid:5
158 a8002006 ->pgnum:1, fpn_num:6, pid:5
159 88004003 ->pgnum:2, fpn_num:3, pid:1
160 88006002 ->pgnum:3, fpn_num:2, pid:1
161 Loaded a process at input/proc/p1s, PID: 6 PRI0: 15
162 ==ALLOC==
163 alloc 1 page
164 write TLB pgnum=1, fpn_num=6, pid=5
```



```
165
166 =print TLB=
167 a8000007 ->pgnum:0, fpn_num:7, pid:5
168     CPU 0: Put process 4 to run queue
169     CPU 2: Put process 2 to run queue
170     CPU 0: Dispatched process 6
171 Time slot 9
172 ==CALC==
173     CPU 1: Put process 3 to run queue
174     CPU 2: Dispatched process 2
175 ==CALC==
176 a8002006 ->pgnum:1, fpn_num:6, pid:5
177 88004003 ->pgnum:2, fpn_num:3, pid:1
178 88006002 ->pgnum:3, fpn_num:2, pid:1
179     CPU 1: Dispatched process 3
180 ==FREE==
181 free TLB pgnum=0, fpn_num=7,pid=5
182
183 =print TLB=
184 a8002006 ->pgnum:1, fpn_num:6, pid:5
185 88004003 ->pgnum:2, fpn_num:3, pid:1
186 88006002 ->pgnum:3, fpn_num:2, pid:1
187     CPU 3: Put process 5 to run queue
188     CPU 3: Dispatched process 4
189 ==CALC==
190 ==FREE==
191 free TLB pgnum=1, fpn_num=6,pid=5
192
193 =print TLB=
194 88004003 ->pgnum:2, fpn_num:3, pid:1
195 88006002 ->pgnum:3, fpn_num:2, pid:1
196 ==CALC==
197 Time slot 10
198 ==CALC==
199 ==CALC==
200     CPU 1: Put process 3 to run queue
201     CPU 0: Put process 6 to run queue
202     CPU 0: Dispatched process 6
203 ==CALC==
204     CPU 1: Dispatched process 3
205 Time slot 11
206 ==FREE==
207     Loaded a process at input/proc/s0, PID: 7 PRI0: 38
208     CPU 2: Put process 2 to run queue
209     CPU 2: Dispatched process 7
210 ==CALC==
211     CPU 3: Put process 4 to run queue
212 ==FREE==
213 Time slot 12
214 ==CALC==
215 ==CALC==
216     CPU 3: Dispatched process 2
217 ==CALC==
218 ==CALC==
219     CPU 1: Processed 3 has finished
220     CPU 1: Dispatched process 4
221 ==CALC==
222     CPU 2: Put process 7 to run queue
223     CPU 2: Dispatched process 7
224 ==CALC==
225 Time slot 13
226     CPU 0: Put process 6 to run queue
```



```
227 CPU 0: Dispatched process 6
228 ==CALC==
229 CPU 3: Put process 2 to run queue
230 Time slot 14
231 ==CALC==
232 ==CALC==
233 CPU 3: Dispatched process 2
234 ==CALC==
235 ==CALC==
236 CPU 3: Processed 2 has finished
237 Time slot 15
238 CPU 0: Put process 6 to run queue
239 CPU 0: Dispatched process 6
240 ==CALC==
241 CPU 3: Dispatched process 5
242 ==FREE==
243 CPU 2: Put process 7 to run queue
244 CPU 2: Dispatched process 7
245 ==CALC==
246 CPU 1: Put process 4 to run queue
247 CPU 1: Dispatched process 4
248 ==CALC==
249 ==ALLOC==
250 Time slot 16
251 alloc 1 page
252 write TLB pgnum=0, fpn_num=7, pid=5
253 Loaded a process at input/proc/s1, PID: 8 PRI0: 0
254 ==CALC==
255 ==CALC==
256
257 =print TLB=
258 a8000007 ->pgnum:0, fpn_num:7, pid:5
259 88004003 ->pgnum:2, fpn_num:3, pid:1
260 88006002 ->pgnum:3, fpn_num:2, pid:1
261 ==CALC==
262 CPU 3: Put process 5 to run queue
263 Time slot 17
264 CPU 0: Put process 6 to run queue
265 CPU 0: Dispatched process 8
266 ==ALLOC==
267 CPU 2: Put process 7 to run queue
268 CPU 2: Dispatched process 7
269 ==CALC==
270 CPU 3: Dispatched process 6
271 ==CALC==
272 CPU 1: Put process 4 to run queue
273 CPU 1: Dispatched process 4
274 ==CALC==
275 alloc 2 page
276 write TLB pgnum=0, fpn_num=0, pid=8
277
278 =print TLB=
279 c0000000 ->pgnum:0, fpn_num:0, pid:8
280 88004003 ->pgnum:2, fpn_num:3, pid:1
281 88006002 ->pgnum:3, fpn_num:2, pid:1
282 write TLB pgnum=1, fpn_num=0, pid=8
283
284 =print TLB=
285 c0000000 ->pgnum:0, fpn_num:0, pid:8
286 c0002000 ->pgnum:1, fpn_num:0, pid:8
287 88004003 ->pgnum:2, fpn_num:3, pid:1
288 88006002 ->pgnum:3, fpn_num:2, pid:1
```



```
289 ==CALC==
290 ==CALC==
291 ==CALC==
292 ==CALC==
293 Time slot 18
294     CPU 3: Processed 6 has finished
295     CPU 3: Dispatched process 5
296     CPU 2: Put process 7 to run queue
297 Time slot 19
298     CPU 1: Put process 4 to run queue
299     CPU 1: Dispatched process 4
300 ==CALC==
301 ==WRITE==
302 write TLB pgnum=1, fpn_num=6, pid=5
303
304 =print TLB=
305 c0000000 ->pgnum:0, fpn_num:0, pid:8
306 a8002006 ->pgnum:1, fpn_num:6, pid:5
307 88004003 ->pgnum:2, fpn_num:3, pid:1
308 88006002 ->pgnum:3, fpn_num:2, pid:1
309     CPU 0: Put process 8 to run queue
310 write data pgn=1, fpn_num=6, pid=5, data=102
311 TLB miss at write region=1 offset=20 value=102
312
313 =print_pgtbl: 0 - 512=
314 00000000: 80000007
315 00000004: 80000006
316
317 =physical memdump=
318 BYTE 00000014: 100
319     CPU 0: Dispatched process 8
320     CPU 2: Dispatched process 7
321 ==CALC==
322 BYTE 000006a4: 102
323 ==CALC==
324 ==WRITE==
325 ==CALC==
326 Time slot 20
327 ==CALC==
328 write TLB pgnum=0, fpn_num=0, pid=5
329
330 =print TLB=
331 a8000000 ->pgnum:0, fpn_num:0, pid:5
332 a8002006 ->pgnum:1, fpn_num:6, pid:5
333 88004003 ->pgnum:2, fpn_num:3, pid:1
334 88006002 ->pgnum:3, fpn_num:2, pid:1
335 ==CALC==
336 write data pgn=0, fpn_num=0, pid=5, data=1
337 TLB miss at write region=2 offset=1000 value=1
338
339 =print_pgtbl: 0 - 512=
340 00000000: c0000000
341 00000004: 80000006
342
343 =physical memdump=
344 BYTE 00000014: 100
345 BYTE 000000b0: 1
346 BYTE 000006a4: 102
347     CPU 3: Put process 5 to run queue
348     CPU 1: Processed 4 has finished
349     CPU 0: Put process 8 to run queue
350     CPU 2: Put process 7 to run queue
```



```
351 Time slot 21
352     CPU 0: Dispatched process 8
353     CPU 1: Dispatched process 1
354 ==READ==
355 TLB miss at read region=1 offset=20
356
357 =print_pgtbl: 0 - 1024=
358 00000000: 80000001
359 00000004: 80000000
360 00000008: 80000003
361 00000012: 80000002
362
363 =physical memdump=
364 BYTE 00000014: 100
365 BYTE 000000b0: 1
366 BYTE 000006a4: 102
367 write TLB pgnum=1, fpn_num=0, pid=1
368
369 =print TLB=
370 a8000000 ->pgnum:0, fpn_num:0, pid:5
371 88002000 ->pgnum:1, fpn_num:0, pid:1
372 88004003 ->pgnum:2, fpn_num:3, pid:1
373 88006002 ->pgnum:3, fpn_num:2, pid:1
374 ==CALC==
375 read data pgn=1, fpn_num=0,pid=1, data=0
376     CPU 3: Dispatched process 5
377 ==WRITE==
378 write data pgn=0, fpn_num=0, pid=5, data=0, phyaddr=0
379 TLB hit at write region=0 offset=0 value=0
380
381     CPU 2: Dispatched process 7
382 ==CALC==
383 =print_pgtbl: 0 - 512=
384 00000000: c0000000
385 00000004: 80000006
386
387 =physical memdump=
388 BYTE 00000014: 100
389 BYTE 000000b0: 1
390 BYTE 000006a4: 102
391     CPU 3: Processed 5 has finished
392 ==CALC==
393 ==CALC==
394 Time slot 22
395     CPU 3 stopped
396 ==WRITE==
397 write TLB pgnum=0, fpn_num=1, pid=1
398
399 =print TLB=
400 88000001 ->pgnum:0, fpn_num:1, pid:1
401 88002000 ->pgnum:1, fpn_num:0, pid:1
402 88004003 ->pgnum:2, fpn_num:3, pid:1
403 88006002 ->pgnum:3, fpn_num:2, pid:1
404 write data pgn=0, fpn_num=1, pid=1, data=103
405 TLB miss at write region=3 offset=20 value=103
406
407 =print_pgtbl: 0 - 1024=
408 00000000: 80000001
409 00000004: 80000000
410 00000008: 80000003
411 00000012: 80000002
412
```



```
413 =physical memdump=
414 BYTE 00000014: 100
415 BYTE 000000b0: 1
416 BYTE 00000114: 103
417 BYTE 000006a4: 102
418     CPU 1: Put process 1 to run queue
419     CPU 1: Dispatched process 1
420 ==READ==
421 TLB hit at read region=3 offset=20
422
423 =print_pgtbl: 0 - 1024=
424 00000000: 80000001
425 00000004: 80000000
426 00000008: 80000003
427 00000012: 80000002
428
429 =physical memdump=
430 BYTE 00000014: 100
431 BYTE 000000b0: 1
432 BYTE 00000114: 103
433 BYTE 000006a4: 102
434 read data pgn=0, fpn_num=1,pid=1, data=103, phyaddr=276
435 Time slot 23
436     CPU 0: Put process 8 to run queue
437     CPU 0: Dispatched process 8
438 ==CALC==
439     CPU 2: Put process 7 to run queue
440     CPU 2: Dispatched process 7
441 ==CALC==
442 ==CALC==
443 Time slot 24
444     CPU 0: Processed 8 has finished
445     CPU 0 stopped
446 ==FREE==
447 free TLB pgnum=2, fpn_num=3,pid=1
448
449 =print TLB=
450 88000001 ->pgnum:0, fpn_num:1, pid:1
451 88002000 ->pgnum:1, fpn_num:0, pid:1
452 88006002 ->pgnum:3, fpn_num:2, pid:1
453 free TLB pgnum=3, fpn_num=2,pid=1
454
455 =print TLB=
456 88000001 ->pgnum:0, fpn_num:1, pid:1
457 88002000 ->pgnum:1, fpn_num:0, pid:1
458     CPU 2: Put process 7 to run queue
459     CPU 2: Dispatched process 7
460 ==CALC==
461 Time slot 25
462     CPU 1: Processed 1 has finished
463     CPU 1 stopped
464 Time slot 26
465     CPU 2: Processed 7 has finished
466     CPU 2 stopped
467
468 Numbers of TLB HIT: 3
469 Numbers of TLB MISS: 4
470 Numbers of TLB access: 7
471 TLB HIT rate: 0.428571
472 TLB MISS rate: 0.571429
473
474 =print TLB=
```

```
475 88000001 ->pgnum:0, fpn_num:1, pid:1
476 88002000 ->pgnum:1, fpn_num:0, pid:1
477
```

4.2 Answer the question.

Question: What will happen if the synchronization is not handled in your simple OS? Illustrate the problem of your simple OS by example.

Answer: A race condition is a situation that occurs in concurrent systems, where the behavior or outcome of the system depends on the relative timing or interleaving of multiple concurrent operations. In other words, it is a race between two or more concurrent threads or processes to access and modify shared resources, leading to unexpected and erroneous results.

Based on the definition of race condition, the resource that is shared among processes is the physical memory. One typical example that when 2 processes run in 2 different CPU access the free frame list of RAM and take out the same frame number.

References

- [1] A. Shapiro, D. Dentcheva, and A. Ruszczyński, Lectures on stochastic programming: modeling and theory. SIAM, 2021.
- [2] L. Wang, “A two-stage stochastic programming framework for evacuation planning in disaster responses,” Computers & Industrial Engineering, vol. 145, p. 106458, 2020.
- [3] S. W. Wallace and W. T. Ziemba, Applications of stochastic programming. SIAM, 2005
- [4] topcoder. MINIMUM COST FLOW PART ONE: KEY CONCEPTS
<https://www.topcoder.com/thrive/articles/Minimum%20Cost%20Flow%20Part%20One:%20Key%20Concepts>
- [5] topcoder. MINIMUM COST FLOW PART TWO: ALGORITHMS
<https://www.topcoder.com/thrive/articles/Minimum%20Cost%20Flow%20Part%20Two:%20Algorithms>