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

Course: Operating Systems

Course Assigment

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

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.

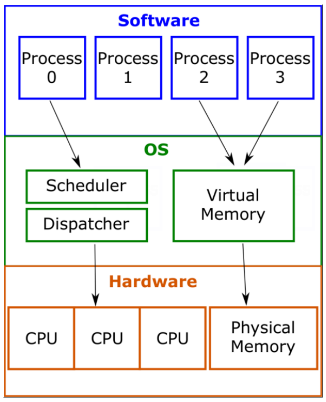


Figure 1: Overview of the modules in a simple operating system

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., slot = (MAX PRIO - prio). When the allocation is exhausted, the system must switch resources to another process in the next queue.

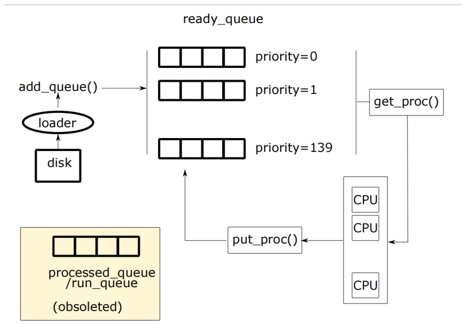


Figure 2: Operation of the Scheduler System

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.

```
void enqueue(struct queue_t * q, struct pcb_t * proc) {
        /* TODO: put a new process to queue [q] */
        int sizeOFqueue = q->size;
  if(sizeOFqueue<MAX_QUEUE_SIZE && sizeOFqueue>=0)
  {
    q->proc[sizeOFqueue]=proc;
    q->size = sizeOFqueue+1;
  }
}

```

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

```
struct pcb_t * dequeue(struct queue_t * q) {
    /* TODO: return a pcb whose prioprity is the highest
     * in the queue [q] and remember to remove it from q
     * */
int sizeOfQueue = q->size;
if( sizeOfQueue ==1 )
{
  struct pcb_t* return_proc = q->proc[0];
  q->proc[0]=NULL;
  q->size =0;
  return return_proc;
}
else if( sizeOfQueue >1 && sizeOfQueue <= MAX_QUEUE_SIZE)
{
  struct pcb_t* return_proc= q->proc[0];// process will be returned
  uint32_t highest_prior = q->proc[0]->priority;
  int position =0;
  int i;
  for(i=0 ; i < sizeOfQueue ;i++)
  {
    if( highest_prior < q->proc[i]->priority )
    {
      highest_prior =q->proc[i]->priority;
      position = i;
    }
  }
  return_proc = q->proc[position];
  if(position == (sizeOfQueue -1 )) // highest at the end of queue
  {
    q->proc[position] =NULL;
    q->size = sizeOfQueue -1;
    return return_proc;
  }
  else
```

```
      {
        for(i = position+1;i< sizeOfQueue; i++)
        {
          q->proc[i-1]=q->proc[i];
        }
        q->proc[sizeOfQueue -1]=NULL;
        q->size = sizeOfQueue -1;
        return return_proc;
      }
  }
  return NULL;
}

```

2.3.2 sched.c

get-mlq-proc function

```
    struct pcb_t * get_mlq_proc(void) {
  struct pcb_t *proc = NULL;
  /*TODO: get a process from PRIORITY [ready_queue].
   * Remember to use lock to protect the queue.
   * */
  pthread_mutex_lock(&queue_lock);
  int i=0;
  int isEmpty =0;
  while(1)
  {
    if (empty(&mlq_ready_queue[i]) || slot[i] == 0)
    {
      if(i >= MAX_PRIO -1){
        if(isEmpty == 1){
          break;
        }
        slot[i] = MAX_PRIO - i;
        i = -1;
        isEmpty++;
        i++;
      }
      slot[i] = MAX_PRIO - i;
      i++;
    }
    else{
    proc = dequeue(&mlq_ready_queue[i]);
    slot[i]--;
    break;
    }
  }
  pthread_mutex_unlock(&queue_lock);
  return proc;
    }

```

get-proc function

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

```c
    pthread_mutex_lock(&queue_lock);
    if (empty(&ready_queue))
    {
      while (!empty(run_queue))
      {
        enqueue(&ready_queue, dequeue(&run_queue));
      }
    }
    proc = dequeue(&ready_queue);
    pthread_mutex_unlock(&queue_lock);

    return proc;
  }

```

2.4 Execution Result

2.4.1 Testcase sched_0

Input:

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

```

Output:

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

```
        CPU 0: Dispatched process  1
Time slot  15
Time slot  16
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  17
Time slot  18
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  19
Time slot  20
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  21
Time slot  22
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  23
        CPU 0: Processed  1 has finished
        CPU 0 stopped

```

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:

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

```

Output:

```
Time slot   0
ld_routine
        Loaded a process at input/proc/s0, PID: 1 PRIO: 4
Time slot   1
        CPU 0: Dispatched process  1
Time slot   2
Time slot   3
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot   4
        Loaded a process at input/proc/s1, PID: 2 PRIO: 0
Time slot   5
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   6
        Loaded a process at input/proc/s2, PID: 3 PRIO: 0
Time slot   7
```

```
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
        Loaded a process at input/proc/s3, PID: 4 PRIO: 0
Time slot   8
Time slot   9
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  2
Time slot  10
Time slot  11
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  12
Time slot  13
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  2
Time slot  14
Time slot  15
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  16
Time slot  17
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  2
Time slot  18
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  3
Time slot  19
Time slot  20
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  3
Time slot  21
Time slot  22
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  3
Time slot  23
Time slot  24
        CPU 0: Processed  3 has finished
        CPU 0: Dispatched process  4
Time slot  25
Time slot  26
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
Time slot  27
Time slot  28
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
Time slot  29
Time slot  30
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
Time slot  31
Time slot  32
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
Time slot  33
Time slot  34
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
Time slot  35
        CPU 0: Processed  4 has finished
        CPU 0: Dispatched process  1
Time slot  36
```

```
Time slot  37
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  38
Time slot  39
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  40
Time slot  41
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  42
Time slot  43
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  44
Time slot  45
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  46
        CPU 0: Processed  1 has finished
        CPU 0 stopped

```

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:

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

```

Output:

```
Time slot   0
ld_routine
        Loaded a process at input/proc/p1s, PID: 1 PRIO: 1
Time slot   1
        CPU 1: Dispatched process  1
        Loaded a process at input/proc/p2s, PID: 2 PRIO: 0
Time slot   2
```

```
        CPU 0: Dispatched process  2
        Loaded a process at input/proc/p3s, PID: 3 PRIO: 0
Time slot   3
Time slot   4
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  3
Time slot   5
Time slot   6
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   7
Time slot   8
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
Time slot   9
Time slot  10
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot  11
Time slot  12
Time slot  13
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
Time slot  14
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  1
Time slot  15
Time slot  16
        CPU 1: Processed  3 has finished
        CPU 1 stopped
Time slot  17
Time slot  18
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  19
Time slot  20
        CPU 0: Processed  1 has finished
        CPU 0 stopped
```

Grant chart:

Proccess/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.

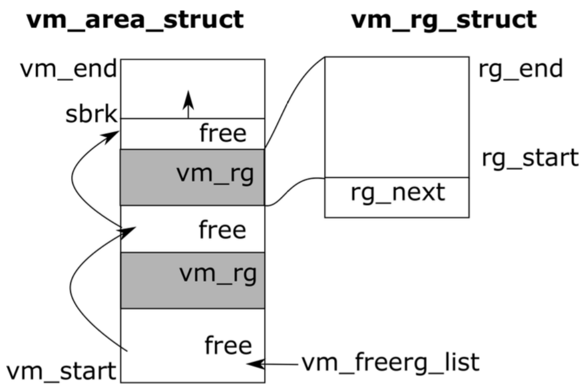


Figure 3: The structure of vm area and region

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.

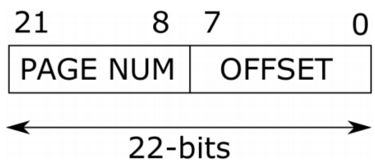


Figure 4: CPU Address

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.

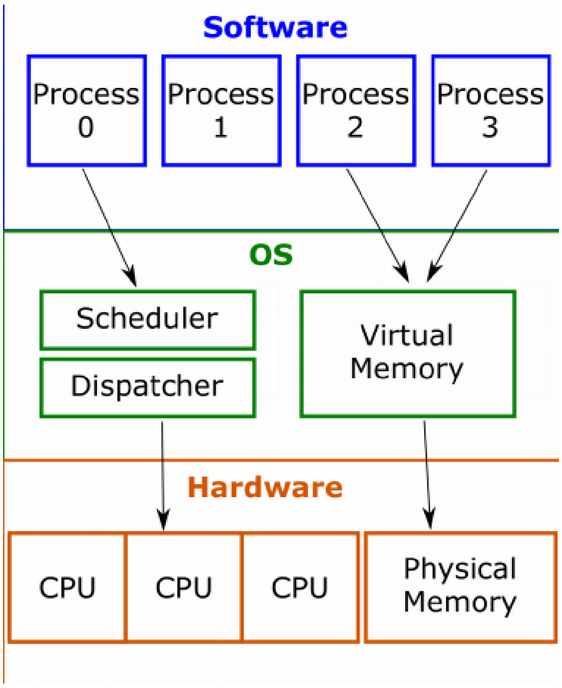


Figure 5: The general view of key modules in this assignment

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

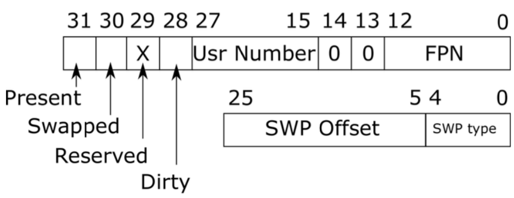


Figure 6: The content of page table entry

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 pf 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.

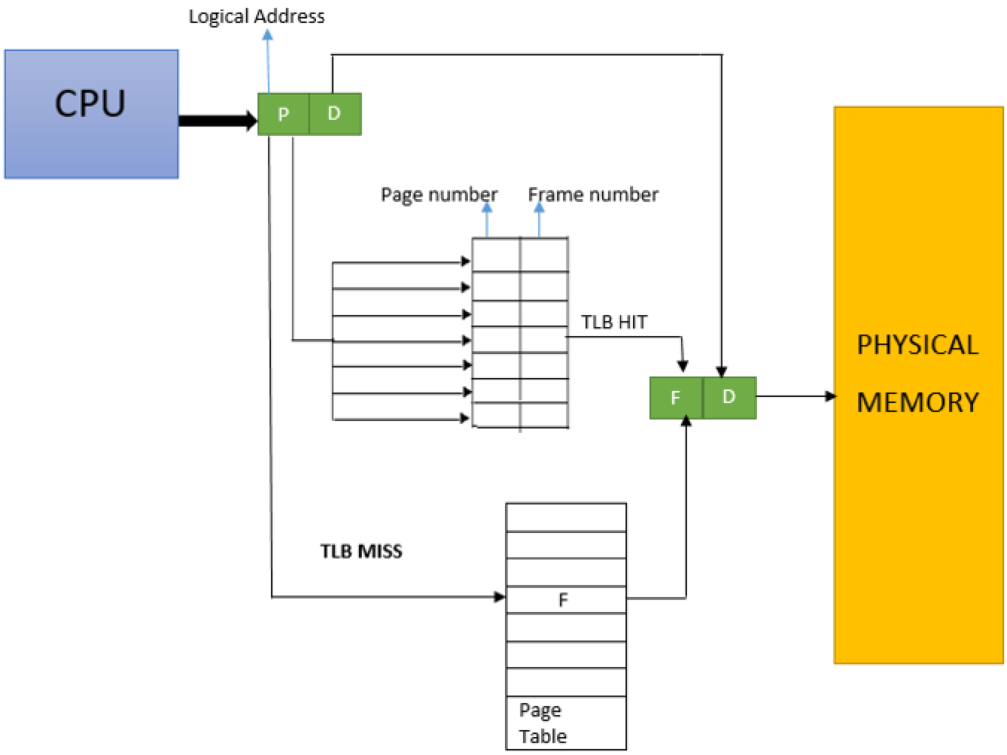
Conclusion: The design of multiple memory segments in the simple operating system enhances mem- ory organization, protection, modularity, and code maintainability, contributing to the overall efficiency and reliability of the operating system.

3.4 Translation Lookaside Buffer (TLB)

3.4.1 Theorical 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

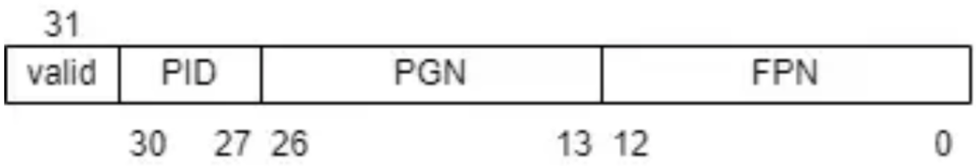


Figure 7: The position of bits in a TLB entry

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

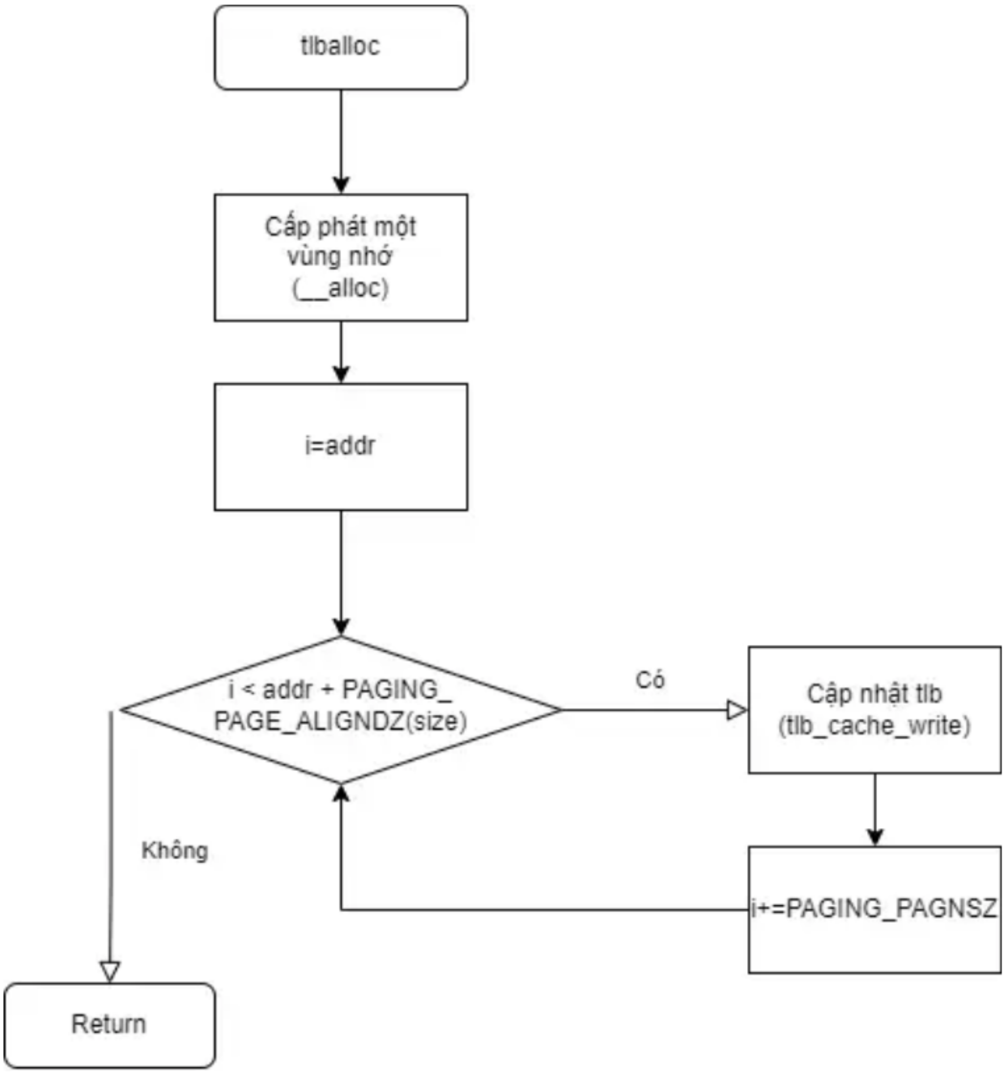


Figure 8: The TLB allocation process

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.

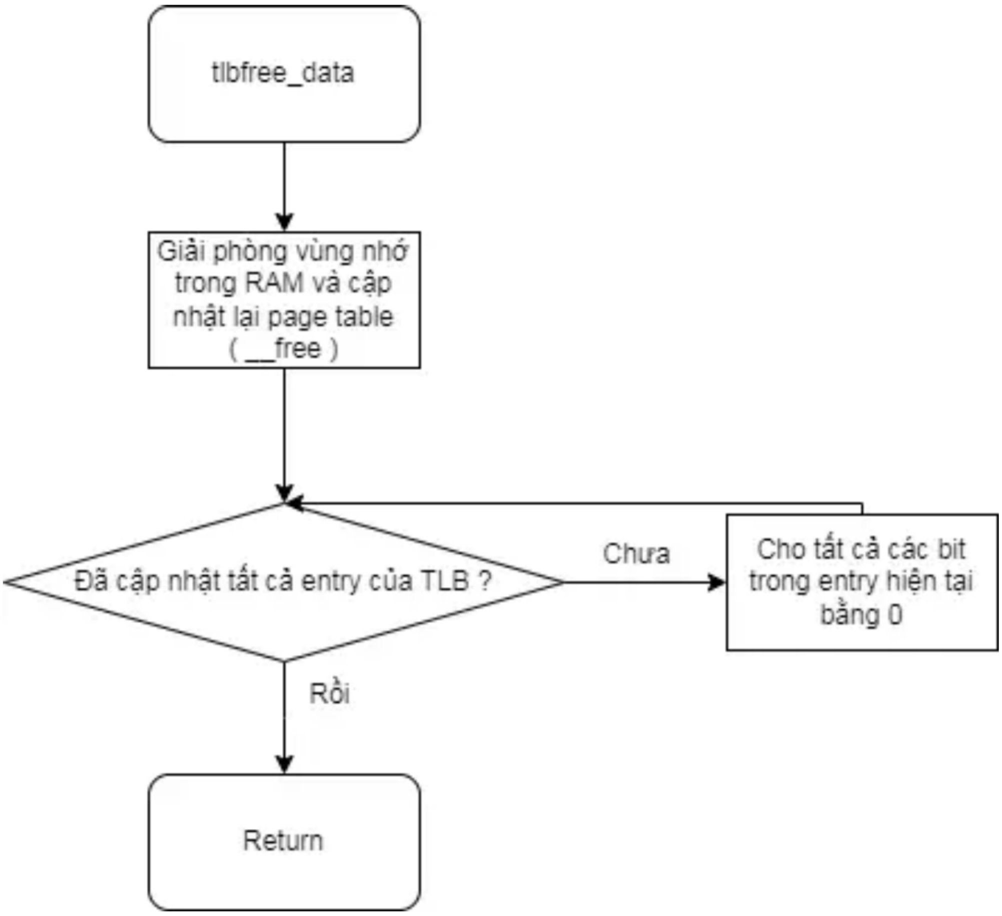


Figure 9: The TLB free process

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.

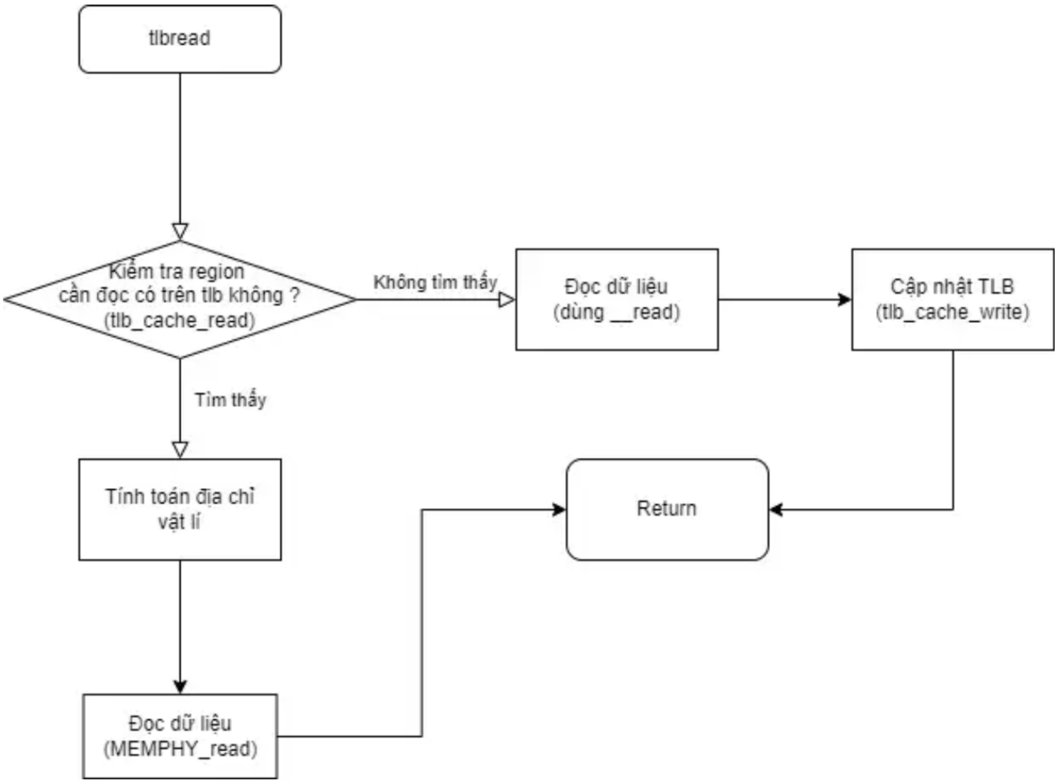


Figure 10: The TLB read process

3.5.5 tlb-write

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

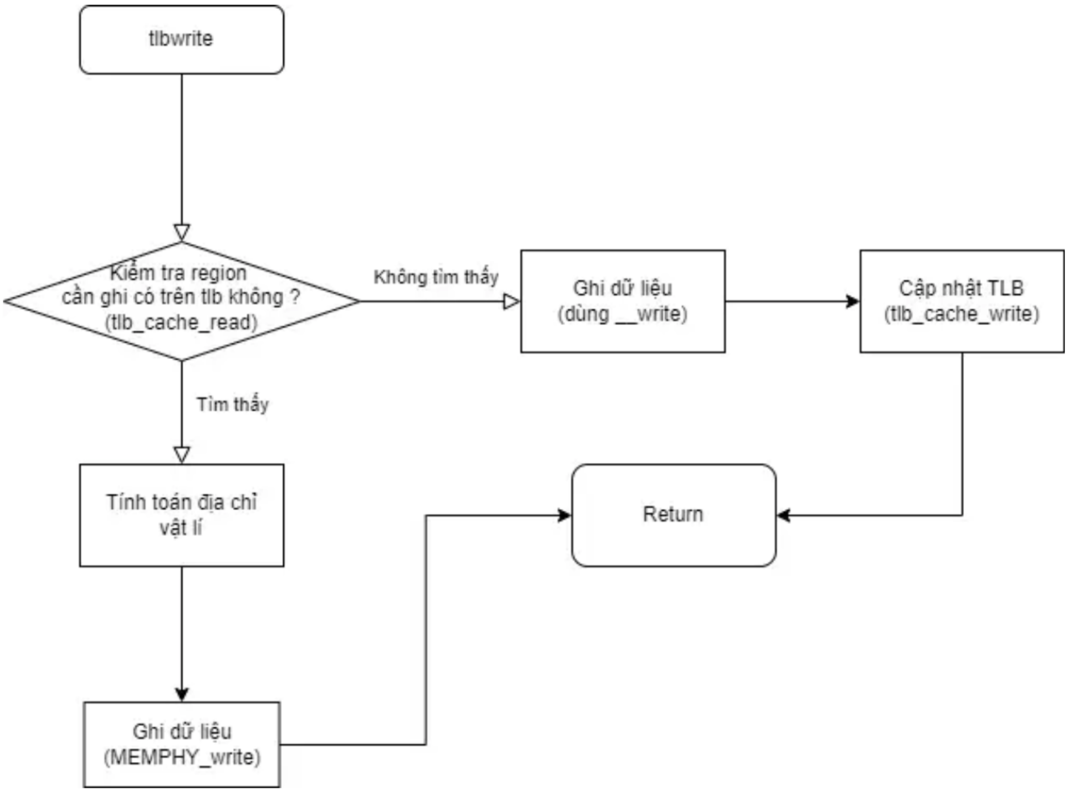


Figure 11: The TLB write process

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.

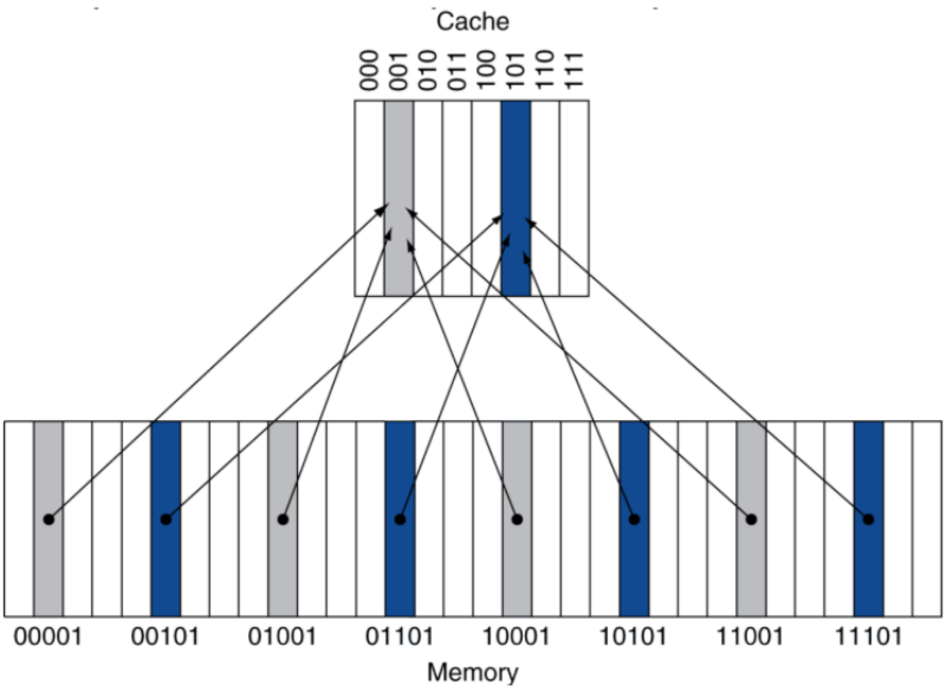


Figure 12: Direct mapping

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

```
  int tlballoc(struct pcb_t *proc, uint32_t size, uint32_t reg_index)
{
  // alloc size region
  int addr, val;
  /* By default using vmaid = 0 */
  val = __alloc(proc, 0, reg_index, size, &addr);
  /* TODO update TLB CACHED frame num of the new allocated page(s)*/
  /* by using tlb_cache_read()/tlb_cache_write()*/
  for (int i = addr; i < addr + PAGING_PAGE_ALIGNSZ(size); i += PAGING_PAGESZ)
  {
    tlb_cache_write(proc->tlb, proc->pid,
                    PAGING_PGN(i), // dung
                    PAGING_FPN_v2(proc->mm->pgd[PAGING_PGN(i)]));
  }
  return val;
}

```

3.7.2 tlb_free_data function

```
  int tlbfree_data(struct pcb_t *proc, uint32_t reg_index)
{
  __free(proc, 0, reg_index);
  /* TODO update TLB CACHED frame num of freed page(s)*/
  /* by using tlb_cache_read()/tlb_cache_write()*/
  struct vm_rg_struct temp = proc->mm->symrgtbl[reg_index]; // co dung ko?
  for (int i = temp.rg_start; i < temp.rg_end; i += PAGING_PAGESZ)
  {
    int pgn = PAGING_PGN(i);
    if (proc->tlb->tlbd[pgn % proc->tlb->maxsz] != 0)
    {
      proc->tlb->tlbd[pgn % proc->tlb->maxsz] = 0;
    }
  }
  return 0;
}

```

3.7.3 tlb_read function

```
  int tlbread(struct pcb_t *proc, uint32_t source,
          uint32_t offset, uint32_t destination)
{

  /* TODO retrieve TLB CACHED frame num of accessing page(s)*/
  /* by using tlb_cache_read()/tlb_cache_write()*/
  /* frmnum is return value of tlb_cache_read/write value*/
  BYTE data;
  int pgnum = PAGING_PGN(proc->mm->symrgtbl[source].rg_start); // ko cong offfset
  uint16_t frnum;
  int return_value = tlb_cache_read(proc->tlb, proc->pid, pgnum, &frnum);
  proc->tlb->total_access++;
#ifdef IODUMP
  if (return_value >= 0)
  {
    printf("TLB hit at read region=%d offset=%d\n",
           source, offset); //?offset
  }
  else
  {
    printf("TLB miss at read region=%d offset=%d\n",
           source, offset);
  }
#ifdef PAGETBL_DUMP
  print_pgtbl(proc, 0, -1); // print max TBL
#endif
  MEMPHY_dump(proc->mram);
#endif
  int val;
  {
    if (return_value >= 0)
    {
      proc->tlb->hit++;
      // MEMPHY read
      int phyaddr = (frnum << PAGING_ADDR_FPN_LOBIT) + offset;
      val = MEMPHY_read(proc->mram, phyaddr, &data);
    }
    else
    {
```

```
      //__read
      val = __read(proc, 0, source, offset, &data);

      tlb_cache_write(proc->tlb, proc->pid, pgnum, PAGING_FPN_v2(proc->mm->pgd[
    pgnum]));
    }
  }
  destination = (uint32_t)data;
  /* TODO update TLB CACHED with frame num of recent accessing page(s)*/
  /* by using tlb_cache_read()/tlb_cache_write()*/
  return val;
}

```

3.7.4 tlb_write fucntion

```
    int tlbwrite(struct pcb_t *proc, BYTE data,
             uint32_t destination, uint32_t offset)
{
  int val;
  //  BYTE frmnum = -1;

  int pgnum = PAGING_PGN(proc->mm->symrgtbl[destination].rg_start);
  uint16_t frnum;
  int return_value = tlb_cache_read(proc->tlb, proc->pid, pgnum, &frnum);
  proc->tlb->total_access++;
  /* TODO retrieve TLB CACHED frame num of accessing page(s))*/
  /* by using tlb_cache_read()/tlb_cache_write()
  frmnum is return value of tlb_cache_read/write value*/
#ifdef IODUMP
  if (return_value >= 0)
    printf("TLB hit at write region=%d offset=%d value=%d\n",
           destination, offset, data);
  else
    printf("TLB miss at write region=%d offset=%d value=%d\n",
           destination, offset, data);
#ifdef PAGETBL_DUMP
  print_pgtbl(proc, 0, -1); // print max TBL
#endif
  MEMPHY_dump(proc->mram);
#endif
  {

    if (return_value >= 0)
    {
      proc->tlb->hit++;
      int phyaddr = (frnum << PAGING_ADDR_FPN_LOBIT) + offset;
      val = MEMPHY_write(proc->mram, phyaddr, data);
    }
    else
    {
      //__write
      val = __write(proc, 0, destination, offset, data);
      tlb_cache_write(proc->tlb, proc->pid, pgnum, PAGING_FPN_v2(proc->mm->pgd[
    pgnum]));
    }
  }
  /* TODO update TLB CACHED with frame num of recent accessing page(s)*/
  /* by using tlb_cache_read()/tlb_cache_write()*/
  return val;
}
```

3.8 Execution Result

3.8.1 Testcase os_0_mlq_paging

Input:

```
6 1 2
1048576 16777216 0 0 0
0 p0s 0
2 p1s 15

```

Output:

```
Time slot   0
ld_routine
        Loaded a process at input/proc/p0s, PID: 1 PRIO: 0
Time slot   1
        CPU 0: Dispatched process  1
==CALC==
Time slot   2
==ALLOC==
        Loaded a process at input/proc/p1s, PID: 2 PRIO: 15
alloc 2 page
write TLB pgnum=0, fpn_num=1, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
write TLB pgnum=1, fpn_num=0, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
Time slot   3
==ALLOC==
alloc 2 page
write TLB pgnum=2, fpn_num=3, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
write TLB pgnum=3, fpn_num=2, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
Time slot   4
==FREE==
free TLB pgnum=0, fpn_num=1,pid=1

=print TLB=
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
free TLB pgnum=1, fpn_num=0,pid=1

```

```
=print TLB=
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
Time slot   5
==ALLOC==
alloc 1 page
write TLB pgnum=1, fpn_num=0, pid=1

=print TLB=
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
Time slot   6
==WRITE==
write data pgn=1, fpn_num=0, pid=1, data=100, phyaddr=20
TLB hit at write region=1 offset=20 value=100

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002

=physical memdump=
BYTE 00000014: 100
Time slot   7
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
==READ==
TLB hit at read region=1 offset=20

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002

=physical memdump=
BYTE 00000014: 100
read data pgn=1, fpn_num=0,pid=1, data=100, phyaddr=20
Time slot   8
==WRITE==
write TLB pgnum=0, fpn_num=1, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
write data pgn=0, fpn_num=1, pid=1, data=103
TLB miss at write region=3 offset=20 value=103

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002

=physical memdump=
BYTE 00000014: 100
BYTE 00000114: 103
Time slot   9
```

```
==READ==
TLB hit at read region=3 offset=20

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002

=physical memdump=
BYTE 00000014: 100
BYTE 00000114: 103
read data pgn=0, fpn_num=1,pid=1, data=103, phyaddr=276
Time slot  10
==FREE==
free TLB pgnum=2, fpn_num=3,pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
free TLB pgnum=3, fpn_num=2,pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
Time slot  11
        CPU 0: Processed  1 has finished
        CPU 0: Dispatched process  2
==CALC==
Time slot  12
==CALC==
Time slot  13
==CALC==
Time slot  14
==CALC==
Time slot  15
==CALC==
Time slot  16
==CALC==
Time slot  17
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
==CALC==
Time slot  18
==CALC==
Time slot  19
==CALC==
Time slot  20
==CALC==
Time slot  21
        CPU 0: Processed  2 has finished
        CPU 0 stopped

Numbers of TLB HIT: 3
Numbers of TLB MISS: 1
Numbers of TLB access: 4
TLB HIT rate: 0.750000
TLB MISS rate: 0.250000

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
```

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

```

3.8.2 Testcase os_1_mlq_paging_small_1K

Input:

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

```

Output:

```
Time slot   0
ld_routine
Time slot   1
        Loaded a process at input/proc/p0s, PID: 1 PRIO: 130
        CPU 3: Dispatched process  1
==CALC==
Time slot   2
        Loaded a process at input/proc/s3, PID: 2 PRIO: 39
==ALLOC==
Time slot   3
        CPU 2: Dispatched process  2
alloc 2 page
write TLB pgnum=0, fpn_num=1, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
==CALC==
write TLB pgnum=1, fpn_num=0, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
==CALC==
        CPU 3: Put process  1 to run queue
        CPU 3: Dispatched process  1
==ALLOC==
alloc 2 page
write TLB pgnum=2, fpn_num=3, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
Time slot   4
write TLB pgnum=3, fpn_num=2, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
```

```
88006002 ->pgnum:3, fpn_num:2, pid:1
        Loaded a process at input/proc/m1s, PID: 3 PRIO: 15
        CPU 2: Put process  2 to run queue
==FREE==
free TLB pgnum=0, fpn_num=1,pid=1
        CPU 2: Dispatched process  2
==CALC==
Time slot   5
        CPU 1: Dispatched process  3
==ALLOC==

=print TLB=
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
alloc 2 page
write TLB pgnum=0, fpn_num=5, pid=3

=print TLB=
98000005 ->pgnum:0, fpn_num:5, pid:3
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
free TLB pgnum=1, fpn_num=0,pid=1

=print TLB=
98000005 ->pgnum:0, fpn_num:5, pid:3
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
write TLB pgnum=1, fpn_num=4, pid=3

=print TLB=
98000005 ->pgnum:0, fpn_num:5, pid:3
98002004 ->pgnum:1, fpn_num:4, pid:3
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        CPU 3: Put process  1 to run queue
        Loaded a process at input/proc/s2, PID: 4 PRIO: 120
==ALLOC==
alloc 1 page
Time slot   6
write TLB pgnum=1, fpn_num=4, pid=3

=print TLB=
98000005 ->pgnum:0, fpn_num:5, pid:3
98002004 ->pgnum:1, fpn_num:4, pid:3
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->     CPU 3: Dispatched process  1
==ALLOC==
alloc 1 page
pgnum:3, fpn_num:2, pid:1
==CALC==
write TLB pgnum=1, fpn_num=0, pid=1

=print TLB=
98000005 ->pgnum:0, fpn_num:5, pid:3
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
        CPU 2: Put process  2 to run queue
```

```
        CPU 2: Dispatched process  2
==CALC==
        CPU 0: Dispatched process  4
==CALC==
==FREE==
==WRITE==
write data pgn=1, fpn_num=0, pid=1, data=100, phyaddr=20
TLB hit at write region=1 offset=20 value=100

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002

=physical memdump=
BYTE 00000014: 100
free TLB pgnum=0, fpn_num=5,pid=3

=print TLB=
Time slot   7
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
free TLB pgnum=1, fpn_num=0,pid=1

=print TLB=
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        Loaded a process at input/proc/m0s, PID: 5 PRIO: 120
        CPU 3: Put process  1 to run queue
==ALLOC==
alloc 1 page
==CALC==
==CALC==
Time slot   8
write TLB pgnum=0, fpn_num=5, pid=3

=print TLB=
98000005 ->pgnum:0, fpn_num:5, pid:3
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        CPU 3: Dispatched process  5
==ALLOC==
alloc 2 page
write TLB pgnum=0, fpn_num=7, pid=5

=print TLB=
a8000007 ->pgnum:0, fpn_num:7, pid:5
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
write TLB pgnum=1, fpn_num=6, pid=5

=print TLB=
a8000007 ->pgnum:0, fpn_num:7, pid:5
a8002006 ->pgnum:1, fpn_num:6, pid:5
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        Loaded a process at input/proc/p1s, PID: 6 PRIO: 15
==ALLOC==
alloc 1 page
write TLB pgnum=1, fpn_num=6, pid=5
```

```

=print TLB=
a8000007 ->pgnum:0, fpn_num:7, pid:5
        CPU 0: Put process  4 to run queue
        CPU 2: Put process  2 to run queue
        CPU 0: Dispatched process  6
Time slot   9
==CALC==
        CPU 1: Put process  3 to run queue
        CPU 2: Dispatched process  2
==CALC==
a8002006 ->pgnum:1, fpn_num:6, pid:5
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        CPU 1: Dispatched process  3
==FREE==
free TLB pgnum=0, fpn_num=7,pid=5

=print TLB=
a8002006 ->pgnum:1, fpn_num:6, pid:5
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        CPU 3: Put process  5 to run queue
        CPU 3: Dispatched process  4
==CALC==
==FREE==
free TLB pgnum=1, fpn_num=6,pid=5

=print TLB=
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
==CALC==
Time slot  10
==CALC==
==CALC==
        CPU 1: Put process  3 to run queue
        CPU 0: Put process  6 to run queue
        CPU 0: Dispatched process  6
==CALC==
        CPU 1: Dispatched process  3
Time slot  11
==FREE==
        Loaded a process at input/proc/s0, PID: 7 PRIO: 38
        CPU 2: Put process  2 to run queue
        CPU 2: Dispatched process  7
==CALC==
        CPU 3: Put process  4 to run queue
==FREE==
Time slot  12
==CALC==
==CALC==
        CPU 3: Dispatched process  2
==CALC==
==CALC==
        CPU 1: Processed  3 has finished
        CPU 1: Dispatched process  4
==CALC==
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
==CALC==
Time slot  13
        CPU 0: Put process  6 to run queue
```

```
        CPU 0: Dispatched process  6
==CALC==
        CPU 3: Put process  2 to run queue
Time slot  14
==CALC==
==CALC==
        CPU 3: Dispatched process  2
==CALC==
==CALC==
        CPU 3: Processed  2 has finished
Time slot  15
        CPU 0: Put process  6 to run queue
        CPU 0: Dispatched process  6
==CALC==
        CPU 3: Dispatched process  5
==FREE==
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
==CALC==
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  4
==CALC==
==ALLOC==
Time slot  16
alloc 1 page
write TLB pgnum=0, fpn_num=7, pid=5
        Loaded a process at input/proc/s1, PID: 8 PRIO: 0
==CALC==
==CALC==

=print TLB=
a8000007 ->pgnum:0, fpn_num:7, pid:5
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
==CALC==
        CPU 3: Put process  5 to run queue
Time slot  17
        CPU 0: Put process  6 to run queue
        CPU 0: Dispatched process  8
==ALLOC==
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
==CALC==
        CPU 3: Dispatched process  6
==CALC==
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  4
==CALC==
alloc 2 page
write TLB pgnum=0, fpn_num=0, pid=8

=print TLB=
c0000000 ->pgnum:0, fpn_num:0, pid:8
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
write TLB pgnum=1, fpn_num=0, pid=8

=print TLB=
c0000000 ->pgnum:0, fpn_num:0, pid:8
c0002000 ->pgnum:1, fpn_num:0, pid:8
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
```

```
==CALC==
==CALC==
==CALC==
==CALC==
Time slot  18
        CPU 3: Processed  6 has finished
        CPU 3: Dispatched process  5
        CPU 2: Put process  7 to run queue
Time slot  19
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  4
==CALC==
==WRITE==
write TLB pgnum=1, fpn_num=6, pid=5

=print TLB=
c0000000 ->pgnum:0, fpn_num:0, pid:8
a8002006 ->pgnum:1, fpn_num:6, pid:5
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        CPU 0: Put process  8 to run queue
write data pgn=1, fpn_num=6, pid=5, data=102
TLB miss at write region=1 offset=20 value=102

=print_pgtbl: 0 - 512=
00000000: 80000007
00000004: 80000006

=physical memdump=
BYTE 00000014: 100
        CPU 0: Dispatched process  8
        CPU 2: Dispatched process  7
==CALC==
BYTE 000006a4: 102
==CALC==
==WRITE==
==CALC==
Time slot  20
==CALC==
write TLB pgnum=0, fpn_num=0, pid=5

=print TLB=
a8000000 ->pgnum:0, fpn_num:0, pid:5
a8002006 ->pgnum:1, fpn_num:6, pid:5
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
==CALC==
write data pgn=0, fpn_num=0, pid=5, data=1
TLB miss at write region=2 offset=1000 value=1

=print_pgtbl: 0 - 512=
00000000: c0000000
00000004: 80000006

=physical memdump=
BYTE 00000014: 100
BYTE 000000b0: 1
BYTE 000006a4: 102
        CPU 3: Put process  5 to run queue
        CPU 1: Processed  4 has finished
        CPU 0: Put process  8 to run queue
        CPU 2: Put process  7 to run queue
```

```
Time slot  21
        CPU 0: Dispatched process  8
        CPU 1: Dispatched process  1
==READ==
TLB miss at read region=1 offset=20

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002

=physical memdump=
BYTE 00000014: 100
BYTE 000000b0: 1
BYTE 000006a4: 102
write TLB pgnum=1, fpn_num=0, pid=1

=print TLB=
a8000000 ->pgnum:0, fpn_num:0, pid:5
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
==CALC==
read data pgn=1, fpn_num=0,pid=1, data=0
        CPU 3: Dispatched process  5
==WRITE==
write data pgn=0, fpn_num=0, pid=5, data=0, phyaddr=0
TLB hit at write region=0 offset=0 value=0

        CPU 2: Dispatched process  7
==CALC==
=print_pgtbl: 0 - 512=
00000000: c0000000
00000004: 80000006

=physical memdump=
BYTE 00000014: 100
BYTE 000000b0: 1
BYTE 000006a4: 102
        CPU 3: Processed  5 has finished
==CALC==
==CALC==
Time slot  22
        CPU 3 stopped
==WRITE==
write TLB pgnum=0, fpn_num=1, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
write data pgn=0, fpn_num=1, pid=1, data=103
TLB miss at write region=3 offset=20 value=103

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002

```

```
=physical memdump=
BYTE 00000014: 100
BYTE 000000b0: 1
BYTE 00000114: 103
BYTE 000006a4: 102
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  1
==READ==
TLB hit at read region=3 offset=20

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002

=physical memdump=
BYTE 00000014: 100
BYTE 000000b0: 1
BYTE 00000114: 103
BYTE 000006a4: 102
read data pgn=0, fpn_num=1,pid=1, data=103, phyaddr=276
Time slot  23
        CPU 0: Put process  8 to run queue
        CPU 0: Dispatched process  8
==CALC==
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
==CALC==
==CALC==
Time slot  24
        CPU 0: Processed  8 has finished
        CPU 0 stopped
==FREE==
free TLB pgnum=2, fpn_num=3,pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
free TLB pgnum=3, fpn_num=2,pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
==CALC==
Time slot  25
        CPU 1: Processed  1 has finished
        CPU 1 stopped
Time slot  26
        CPU 2: Processed  7 has finished
        CPU 2 stopped

Numbers of TLB HIT: 3
Numbers of TLB MISS: 4
Numbers of TLB access: 7
TLB HIT rate: 0.428571
TLB MISS rate: 0.571429

=print TLB=
```

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

```

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:

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

```
11 s0 38
16 s1 0

```

4.1.2.a Without synchronization:

```
Time slot   0
ld_routine
        Loaded a process at input/proc/p0s, PID: 1 PRIO: 130
Time slot   1
        CPU 3: Dispatched process  1
==CALC==
Time slot   2
        Loaded a process at input/proc/s3, PID: 2 PRIO: 39
==ALLOC==
        CPU 2: Dispatched process  2
alloc 2 page
write TLB pgnum=0, fpn_num=1, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
==CALC==
Time slot   3
write TLB pgnum=1, fpn_num=0, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
        Loaded a process at input/proc/m1s, PID: 3 PRIO: 15
Time slot   4
==CALC==
        CPU 1: Dispatched process  3
==ALLOC==
        CPU 3: Put process  1 to run queue
alloc 2 page
        CPU 3: Dispatched process  1
==ALLOC==
write TLB pgnum=0, fpn_num=3, pid=3

=print TLB=
98000003 ->pgnum:0, fpn_num:3, pid:3
88002000 ->pgnum:1, fpn_num:0, pid:1
alloc 2 page
write TLB pgnum=2, fpn_num=5, pid=1

=print TLB=
98000003 ->pgnum:0, fpn_num:3, pid:3
88002000 ->pgnum:1, fpn_num:0, pid:1
88004005 ->pgnum:2, fpn_num:5, pid:1
write TLB pgnum=1, fpn_num=2, pid=3

=print TLB=
98000003 ->pgnum:0, fpn_num:3, pid:3
98002002 ->pgnum:1, fpn_num:2, pid:3
88004005 ->pgnum:2, fpn_num:5, pid:1
write TLB pgnum=3, fpn_num=4, pid=1

=print TLB=
98000003 ->pgnum:0, fpn_num:3, pid:3
98002002 ->pgnum:1, fpn_num:2, pid:3
88004005 ->pgnum:2, fpn_num:5, pid:1
```

```
88006004 ->pgnum:3, fpn_num:4, pid:1
==FREE==
        CPU 2: Put process  2 to run queue
Time slot   5
==ALLOC==
alloc 1 page
write TLB pgnum=1, fpn_num=2, pid=3

=print TLB=
98000003 ->     CPU 2: Dispatched process  2
==CALC==
pgnum:0, fpn_num:3, pid:3
98002002 ->pgnum:1, fpn_num:2, pid:3
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
free TLB pgnum=0, fpn_num=3,pid=3

=print TLB=
98002002 ->pgnum:1, fpn_num:2, pid:3
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
free TLB pgnum=1, fpn_num=2,pid=3

=print TLB=
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
        CPU 3: Put process  1 to run queue
Time slot   6
        Loaded a process at input/proc/s2, PID: 4 PRIO: 120
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
==FREE==
==CALC==
        CPU 3: Dispatched process  1
==ALLOC==
alloc 1 page
write TLB pgnum=1, fpn_num=0, pid=1

=print TLB=
88002000 ->pgnum:1, fpn_num:0, pid:1
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
        CPU 0: Dispatched process  1
==WRITE==
write data pgn=1, fpn_num=0, pid=1, data=100, phyaddr=20
TLB hit at write region=1 offset=20 value=100

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000005
00000012: 80000004

=physical memdump=
BYTE 00000014: 100
==READ==
Time slot   7
==ALLOC==
alloc 1 page
write TLB pgnum=0, fpn_num=3, pid=3

=print TLB=
```

```
98000003 ->pgnum:0, fpn_num:3, pid:3
88002000 ->pgnum:1, fpn_num:0, pid:1
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
==WRITE==
write TLB pgnum=0, fpn_num=1, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
TLB hit at read region=1 offset=20

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000005
00000012: 80000004

=physical memdump=
BYTE 00000014: 100
BYTE 00000114: 103
read data pgn=1, fpn_num=0,pid=1, data=100, phyaddr=20
        CPU 2: Put process  2 to run queue
        CPU 2: Dispatched process  2
==CALC==
write data pgn=0, fpn_num=1, pid=1, data=103
TLB miss at write region=3 offset=20 value=103

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000005
00000012: 80000004

=physical memdump=
BYTE 00000014: 100
BYTE 00000114: 103
        Loaded a process at input/proc/m0s, PID: 5 PRIO: 120
        CPU 3: Put process  1 to run queue
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
==FREE==
==CALC==
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  5
==ALLOC==
free TLB pgnum=0, fpn_num=1,pid=1

=print TLB=
88002000 ->pgnum:1, fpn_num:0, pid:1
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
alloc 2 page
Time slot   8
        CPU 3: Dispatched process  4
write TLB pgnum=0, fpn_num=7, pid=5

=print TLB=
a8000007 ->pgnum:0, fpn_num:7, pid:5
88002000 ->pgnum:1, fpn_num:0, pid:1
```

```
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
==CALC==
write TLB pgnum=1, fpn_num=6, pid=5

=print TLB=
a8000007 ->pgnum:0, fpn_num:7, pid:5
a8002006 ->pgnum:1, fpn_num:6, pid:5
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
        Loaded a process at input/proc/p1s, PID: 6 PRIO: 15
Time slot   9
==ALLOC==
alloc 1 page
        CPU 2: Put process  2 to run queue
        CPU 2: Dispatched process  6
==CALC==
write TLB pgnum=1, fpn_num=6, pid=5

=print TLB=
a8000007 ->pgnum:0, fpn_num:7, pid:5
a8002006 ->pgnum:1, fpn_num:6, pid:5
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
==FREE==
free TLB pgnum=1, fpn_num=6,pid=5

=print TLB=
a8000007 ->pgnum:0, fpn_num:7, pid:5
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
==CALC==
        CPU 3: Put process  4 to run queue
        CPU 3: Dispatched process  2
==CALC==
        CPU 1: Put process  3 to run queue
Time slot  10
        CPU 0: Put process  5 to run queue
        CPU 0: Dispatched process  4
==CALC==
        CPU 1: Dispatched process  3
==FREE==
==CALC==
        Loaded a process at input/proc/s0, PID: 7 PRIO: 38
Time slot  11
        CPU 2: Put process  6 to run queue
        CPU 2: Dispatched process  6
==CALC==
==CALC==
==FREE==
==CALC==
==CALC==
        CPU 3: Put process  2 to run queue
        CPU 3: Dispatched process  7
==CALC==
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
==CALC==
        CPU 1: Processed  3 has finished
        CPU 1: Dispatched process  4
==CALC==
Time slot  12
```

```
==CALC==
==CALC==
==CALC==
Time slot  13
        CPU 2: Put process  6 to run queue
        CPU 2: Dispatched process  6
==CALC==
==CALC==
        CPU 1: Put process  4 to run queue
        CPU 3: Put process  7 to run queue
        CPU 3: Dispatched process  7
==CALC==
        CPU 1: Dispatched process  4
==CALC==
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
==CALC==
Time slot  14
==CALC==
==CALC==
Time slot  15
        CPU 2: Put process  6 to run queue
        CPU 2: Dispatched process  6
==CALC==
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  5
==FREE==
free TLB pgnum=0, fpn_num=7,pid=5

=print TLB=
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
        CPU 3: Put process  7 to run queue
        CPU 3: Dispatched process  7
==CALC==
Time slot  16
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  4
==CALC==
==ALLOC==
alloc 1 page
write TLB pgnum=0, fpn_num=7, pid=5

=print TLB=
a8000007 ->pgnum:0, fpn_num:7, pid:5
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
        Loaded a process at input/proc/s1, PID: 8 PRIO: 0
==CALC==
==CALC==
==CALC==
Time slot  17
        CPU 0: Put process  5 to run queue
        CPU 0: Dispatched process  8
==ALLOC==
alloc 2 page
write TLB pgnum=0, fpn_num=0, pid=8

=print TLB=
c0000000 ->pgnum:0, fpn_num:0, pid:8
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
```

```
        CPU 2: Put process  6 to run queue
        CPU 2: Dispatched process  6
write TLB pgnum=1, fpn_num=0, pid=8

=print TLB=
c0000000 ->pgnum:0, fpn_num:0, pid:8
c0002000 ->pgnum:1, fpn_num:0, pid:8
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
==CALC==
        CPU 3: Put process  7 to run queue
Time slot  18
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  4
==CALC==
==CALC==
        CPU 3: Dispatched process  7
==CALC==
==CALC==
==CALC==
==CALC==
Time slot  19
        CPU 0: Put process  8 to run queue
        CPU 0: Dispatched process  8
==CALC==
        CPU 2: Processed  6 has finished
        CPU 2: Dispatched process  5
==WRITE==
write TLB pgnum=1, fpn_num=6, pid=5

=print TLB=
c0000000 ->pgnum:0, fpn_num:0, pid:8
a8002006 ->pgnum:1, fpn_num:6, pid:5
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
write data pgn=1, fpn_num=6, pid=5, data=102
TLB miss at write region=1 offset=20 value=102

=print_pgtbl: 0 - 512=
00000000: 80000007
00000004: 80000006

=physical memdump=
BYTE 00000014: 100
BYTE 00000114: 103
BYTE 000006a4: 102
        CPU 3: Put process  7 to run queue
        CPU 1: Processed  4 has finished
==CALC==
Time slot  20
==WRITE==
write TLB pgnum=0, fpn_num=0, pid=5

=print TLB=
        CPU 1: Dispatched process  1
==READ==
TLB miss at read region=3 offset=20

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000005
```

```
00000012: 80000004

=physical memdump=
BYTE 00000014: 100
BYTE 000000b0: 1
BYTE 00000114: 103
BYTE 000006a4: 102
write TLB pgnum=0, fpn_num=1, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
a8002006 ->pgnum:1, fpn_num:6, pid:5
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
        CPU 3: Dispatched process  7
==CALC==
read data pgn=0, fpn_num=1,pid=1, data=103
a8000000 ->pgnum:0, fpn_num:1, pid:1
a8002006 ->pgnum:1, fpn_num:6, pid:5
88004005 ->pgnum:2, fpn_num:5, pid:1
88006004 ->pgnum:3, fpn_num:4, pid:1
write data pgn=0, fpn_num=1, pid=1, data=1
TLB miss at write region=2 offset=1000 value=1

=print_pgtbl: 0 - 512=
00000000: c0000000
00000004: 80000006

=physical memdump=
BYTE 00000014: 100
BYTE 000000b0: 1
BYTE 00000114: 103
BYTE 000006a4: 102
==CALC==
==FREE==
free TLB pgnum=2, fpn_num=5,pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
a8002006 ->pgnum:1, fpn_num:6, pid:5
88006004 ->pgnum:3, fpn_num:4, pid:1
Time slot  21
free TLB pgnum=3, fpn_num=4,pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
a8002006 ->pgnum:1, fpn_num:6, pid:5
        CPU 0: Put process  8 to run queue
        CPU 0: Dispatched process  8
==CALC==
        CPU 2: Put process  5 to run queue
        CPU 2: Dispatched process  5
==WRITE==
write TLB pgnum=0, fpn_num=0, pid=5

=print TLB=
a8000000 ->pgnum:0, fpn_num:0, pid:5
a8002006 ->pgnum:1, fpn_num:6, pid:5
write data pgn=0, fpn_num=0, pid=5, data=0
TLB miss at write region=0 offset=0 value=0

=print_pgtbl: 0 - 512=
```

```
00000000: c0000000
00000004: 80000006

=physical memdump=
BYTE 00000014: 100
BYTE 000000b0: 1
BYTE 00000114: 103
BYTE 000006a4: 102
        CPU 3: Put process  7 to run queue
        CPU 1: Processed  1 has finished
Time slot  22
        CPU 3: Dispatched process  7
==CALC==
        CPU 1: Dispatched process 1542034800
==CALC==
        CPU 2: Processed  5 has finished
        CPU 2 stopped
Segmentation fault
```

4.1.2.b With synchronization:

```
Time slot   0
ld_routine
Time slot   1
        Loaded a process at input/proc/p0s, PID: 1 PRIO: 130
        CPU 3: Dispatched process  1
==CALC==
Time slot   2
        Loaded a process at input/proc/s3, PID: 2 PRIO: 39
==ALLOC==
Time slot   3
        CPU 2: Dispatched process  2
alloc 2 page
write TLB pgnum=0, fpn_num=1, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
==CALC==
write TLB pgnum=1, fpn_num=0, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
==CALC==
        CPU 3: Put process  1 to run queue
        CPU 3: Dispatched process  1
==ALLOC==
alloc 2 page
write TLB pgnum=2, fpn_num=3, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
Time slot   4
write TLB pgnum=3, fpn_num=2, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
```

```
88006002 ->pgnum:3, fpn_num:2, pid:1
        Loaded a process at input/proc/m1s, PID: 3 PRIO: 15
        CPU 2: Put process  2 to run queue
==FREE==
free TLB pgnum=0, fpn_num=1,pid=1
        CPU 2: Dispatched process  2
==CALC==
Time slot   5
        CPU 1: Dispatched process  3
==ALLOC==

=print TLB=
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
alloc 2 page
write TLB pgnum=0, fpn_num=5, pid=3

=print TLB=
98000005 ->pgnum:0, fpn_num:5, pid:3
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
free TLB pgnum=1, fpn_num=0,pid=1

=print TLB=
98000005 ->pgnum:0, fpn_num:5, pid:3
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
write TLB pgnum=1, fpn_num=4, pid=3

=print TLB=
98000005 ->pgnum:0, fpn_num:5, pid:3
98002004 ->pgnum:1, fpn_num:4, pid:3
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        CPU 3: Put process  1 to run queue
        Loaded a process at input/proc/s2, PID: 4 PRIO: 120
==ALLOC==
alloc 1 page
Time slot   6
write TLB pgnum=1, fpn_num=4, pid=3

=print TLB=
98000005 ->pgnum:0, fpn_num:5, pid:3
98002004 ->pgnum:1, fpn_num:4, pid:3
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->     CPU 3: Dispatched process  1
==ALLOC==
alloc 1 page
pgnum:3, fpn_num:2, pid:1
==CALC==
write TLB pgnum=1, fpn_num=0, pid=1

=print TLB=
98000005 ->pgnum:0, fpn_num:5, pid:3
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  3
        CPU 2: Put process  2 to run queue
```

```
        CPU 2: Dispatched process  2
==CALC==
        CPU 0: Dispatched process  4
==CALC==
==FREE==
==WRITE==
write data pgn=1, fpn_num=0, pid=1, data=100, phyaddr=20
TLB hit at write region=1 offset=20 value=100

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002

=physical memdump=
BYTE 00000014: 100
free TLB pgnum=0, fpn_num=5,pid=3

=print TLB=
Time slot   7
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
free TLB pgnum=1, fpn_num=0,pid=1

=print TLB=
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        Loaded a process at input/proc/m0s, PID: 5 PRIO: 120
        CPU 3: Put process  1 to run queue
==ALLOC==
alloc 1 page
==CALC==
==CALC==
Time slot   8
write TLB pgnum=0, fpn_num=5, pid=3

=print TLB=
98000005 ->pgnum:0, fpn_num:5, pid:3
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        CPU 3: Dispatched process  5
==ALLOC==
alloc 2 page
write TLB pgnum=0, fpn_num=7, pid=5

=print TLB=
a8000007 ->pgnum:0, fpn_num:7, pid:5
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
write TLB pgnum=1, fpn_num=6, pid=5

=print TLB=
a8000007 ->pgnum:0, fpn_num:7, pid:5
a8002006 ->pgnum:1, fpn_num:6, pid:5
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        Loaded a process at input/proc/p1s, PID: 6 PRIO: 15
==ALLOC==
alloc 1 page
write TLB pgnum=1, fpn_num=6, pid=5
```

```

=print TLB=
a8000007 ->pgnum:0, fpn_num:7, pid:5
        CPU 0: Put process  4 to run queue
        CPU 2: Put process  2 to run queue
        CPU 0: Dispatched process  6
Time slot   9
==CALC==
        CPU 1: Put process  3 to run queue
        CPU 2: Dispatched process  2
==CALC==
a8002006 ->pgnum:1, fpn_num:6, pid:5
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        CPU 1: Dispatched process  3
==FREE==
free TLB pgnum=0, fpn_num=7,pid=5

=print TLB=
a8002006 ->pgnum:1, fpn_num:6, pid:5
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        CPU 3: Put process  5 to run queue
        CPU 3: Dispatched process  4
==CALC==
==FREE==
free TLB pgnum=1, fpn_num=6,pid=5

=print TLB=
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
==CALC==
Time slot  10
==CALC==
==CALC==
        CPU 1: Put process  3 to run queue
        CPU 0: Put process  6 to run queue
        CPU 0: Dispatched process  6
==CALC==
        CPU 1: Dispatched process  3
Time slot  11
==FREE==
        Loaded a process at input/proc/s0, PID: 7 PRIO: 38
        CPU 2: Put process  2 to run queue
        CPU 2: Dispatched process  7
==CALC==
        CPU 3: Put process  4 to run queue
==FREE==
Time slot  12
==CALC==
==CALC==
        CPU 3: Dispatched process  2
==CALC==
==CALC==
        CPU 1: Processed  3 has finished
        CPU 1: Dispatched process  4
==CALC==
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
==CALC==
Time slot  13
        CPU 0: Put process  6 to run queue
```

```
        CPU 0: Dispatched process  6
==CALC==
        CPU 3: Put process  2 to run queue
Time slot  14
==CALC==
==CALC==
        CPU 3: Dispatched process  2
==CALC==
==CALC==
        CPU 3: Processed  2 has finished
Time slot  15
        CPU 0: Put process  6 to run queue
        CPU 0: Dispatched process  6
==CALC==
        CPU 3: Dispatched process  5
==FREE==
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
==CALC==
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  4
==CALC==
==ALLOC==
Time slot  16
alloc 1 page
write TLB pgnum=0, fpn_num=7, pid=5
        Loaded a process at input/proc/s1, PID: 8 PRIO: 0
==CALC==
==CALC==

=print TLB=
a8000007 ->pgnum:0, fpn_num:7, pid:5
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
==CALC==
        CPU 3: Put process  5 to run queue
Time slot  17
        CPU 0: Put process  6 to run queue
        CPU 0: Dispatched process  8
==ALLOC==
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
==CALC==
        CPU 3: Dispatched process  6
==CALC==
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  4
==CALC==
alloc 2 page
write TLB pgnum=0, fpn_num=0, pid=8

=print TLB=
c0000000 ->pgnum:0, fpn_num:0, pid:8
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
write TLB pgnum=1, fpn_num=0, pid=8

=print TLB=
c0000000 ->pgnum:0, fpn_num:0, pid:8
c0002000 ->pgnum:1, fpn_num:0, pid:8
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
```

```
==CALC==
==CALC==
==CALC==
==CALC==
Time slot  18
        CPU 3: Processed  6 has finished
        CPU 3: Dispatched process  5
        CPU 2: Put process  7 to run queue
Time slot  19
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  4
==CALC==
==WRITE==
write TLB pgnum=1, fpn_num=6, pid=5

=print TLB=
c0000000 ->pgnum:0, fpn_num:0, pid:8
a8002006 ->pgnum:1, fpn_num:6, pid:5
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
        CPU 0: Put process  8 to run queue
write data pgn=1, fpn_num=6, pid=5, data=102
TLB miss at write region=1 offset=20 value=102

=print_pgtbl: 0 - 512=
00000000: 80000007
00000004: 80000006

=physical memdump=
BYTE 00000014: 100
        CPU 0: Dispatched process  8
        CPU 2: Dispatched process  7
==CALC==
BYTE 000006a4: 102
==CALC==
==WRITE==
==CALC==
Time slot  20
==CALC==
write TLB pgnum=0, fpn_num=0, pid=5

=print TLB=
a8000000 ->pgnum:0, fpn_num:0, pid:5
a8002006 ->pgnum:1, fpn_num:6, pid:5
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
==CALC==
write data pgn=0, fpn_num=0, pid=5, data=1
TLB miss at write region=2 offset=1000 value=1

=print_pgtbl: 0 - 512=
00000000: c0000000
00000004: 80000006

=physical memdump=
BYTE 00000014: 100
BYTE 000000b0: 1
BYTE 000006a4: 102
        CPU 3: Put process  5 to run queue
        CPU 1: Processed  4 has finished
        CPU 0: Put process  8 to run queue
        CPU 2: Put process  7 to run queue
```

```
Time slot  21
        CPU 0: Dispatched process  8
        CPU 1: Dispatched process  1
==READ==
TLB miss at read region=1 offset=20

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002

=physical memdump=
BYTE 00000014: 100
BYTE 000000b0: 1
BYTE 000006a4: 102
write TLB pgnum=1, fpn_num=0, pid=1

=print TLB=
a8000000 ->pgnum:0, fpn_num:0, pid:5
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
==CALC==
read data pgn=1, fpn_num=0,pid=1, data=0
        CPU 3: Dispatched process  5
==WRITE==
write data pgn=0, fpn_num=0, pid=5, data=0, phyaddr=0
TLB hit at write region=0 offset=0 value=0

        CPU 2: Dispatched process  7
==CALC==
=print_pgtbl: 0 - 512=
00000000: c0000000
00000004: 80000006

=physical memdump=
BYTE 00000014: 100
BYTE 000000b0: 1
BYTE 000006a4: 102
        CPU 3: Processed  5 has finished
==CALC==
==CALC==
Time slot  22
        CPU 3 stopped
==WRITE==
write TLB pgnum=0, fpn_num=1, pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
88004003 ->pgnum:2, fpn_num:3, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
write data pgn=0, fpn_num=1, pid=1, data=103
TLB miss at write region=3 offset=20 value=103

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002

```

```
=physical memdump=
BYTE 00000014: 100
BYTE 000000b0: 1
BYTE 00000114: 103
BYTE 000006a4: 102
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  1
==READ==
TLB hit at read region=3 offset=20

=print_pgtbl: 0 - 1024=
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002

=physical memdump=
BYTE 00000014: 100
BYTE 000000b0: 1
BYTE 00000114: 103
BYTE 000006a4: 102
read data pgn=0, fpn_num=1,pid=1, data=103, phyaddr=276
Time slot  23
        CPU 0: Put process  8 to run queue
        CPU 0: Dispatched process  8
==CALC==
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
==CALC==
==CALC==
Time slot  24
        CPU 0: Processed  8 has finished
        CPU 0 stopped
==FREE==
free TLB pgnum=2, fpn_num=3,pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
88006002 ->pgnum:3, fpn_num:2, pid:1
free TLB pgnum=3, fpn_num=2,pid=1

=print TLB=
88000001 ->pgnum:0, fpn_num:1, pid:1
88002000 ->pgnum:1, fpn_num:0, pid:1
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
==CALC==
Time slot  25
        CPU 1: Processed  1 has finished
        CPU 1 stopped
Time slot  26
        CPU 2: Processed  7 has finished
        CPU 2 stopped

Numbers of TLB HIT: 3
Numbers of TLB MISS: 4
Numbers of TLB access: 7
TLB HIT rate: 0.428571
TLB MISS rate: 0.571429

=print TLB=
```

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

```

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. Ruszczynski, 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