Operating System (CSE231)

Section-B

Quiz 2

Total Marks: 30 **Time: 40 mins**

---

1. (7 points) Answer the following questions.

   (a) (2 points) An MLFQ scheduler uses a high-priority queue (Q1) with a 10 ms quantum and a low-priority queue (Q0). A job is demoted from Q1 to Q0 if it uses its entire quantum. Job A (runtime 25 ms) arrives at T=0, and Job B (runtime 8 ms) arrives at T=2. What is the completion time for Job B?
   T = 0 to 10 ms: Job A runs from the high-priority queue, Q1. During this time, Job B arrives at T=2 and waits in Q1. At T=10, Job A has used its full quantum and is demoted to Q0;
   T = 10 to 18 ms: The scheduler now sees Job B in the high-priority queue (Q1) and Job A in the low-priority queue (Q0). It executes Job B. Job B runs for its entire 8 ms burst.
   T = 18 ms: Job B finishes its execution.
   The completion time of Job B is **18 ms**.

   (b) (2 points) An MLFQ scheduler has two queues: Q1 (highest, 8 ms quantum) and Q0 (lowest, 16 ms quantum). A job moves from Q1 to Q0 only if it uses its entire 8 ms quantum. A job in Q0 is preempted by any new arrival in Q1. **Job A (runtime 20 ms)** arrives at T=0; **Job B (runtime 12 ms)** arrives at T=4; **Job C (runtime 6 ms)** arrives at T=12.
   What is the average turnaround time for these three jobs?
   T=0ms: Job A arrives and starts executing in the high-priority queue (Q1).
   T=4ms: Job B arrives and is placed in the Q1 ready queue. Job A continues to execute.
   T=8ms: Job A has used its entire 8 ms quantum. It is preempted and moved to the low-priority queue (Q0). Job B begins executing from Q1.
   T=12ms: Job C arrives and is placed in the Q1 ready queue. Job B continues to execute.
   T=16ms: Job B has used its entire 8 ms quantum (from T=8 to T=16). It is preempted and moved to Q0. Job C begins executing from Q1.
   T=22ms: Job C finishes execution after 6 ms, which is less than its quantum. Job C completes. The scheduler now checks Q0. Job A is selected to run (FCFS within the queue).
   T=34ms: Job A runs for its remaining 12 ms (its Q0 quantum is 16 ms, so it completes without preemption). Job A completes. Job B begins executing.
   T=38ms: Job B runs for its remaining 4 ms. Job B completes.
   Turnaround Times: Job A: 34 ms - 0 ms = 34 ms; Job B: 38 ms - 4 ms = 34 ms; Job C: 22 ms - 12 ms = 10 ms; The average turnaround time is (34 + 34 + 10) / 3 = **26 ms**.

   (c) (1 point) What mechanism in MLFQ prevents starvation of low-priority processes? Priority Boost

   (d) (1 point) MLFQ gives higher priority to what type of jobs? Short (or I/O-bound) jobs.

(e) (1 point) A process P4 is currently in Queue 1 (Q1), which has a time slice of 40ms. Queue 0 is the lower priority queue. The scheduler runs P4, and after 15ms, P4 executes a system call to read from a disk, causing it to block. Based on the standard rules of MLFQ, when the disk read is complete and P4 is ready to run again, it will be placed back into Queue 1.

2. (5 points) Below are the C codes involving different scenarios. Assuming there are no syntax or semantic errors, answer the questions:

(a) (2 points) Given the following C code, what will be its output?

```
#include <stdio.h>
// Queue 0 is the highest priority, Queue 2 is the lowest.
void update_process_queue(int process_id, int current_queue,
int time_slice, int time_used) {
    int new_queue = current_queue;
    if (time_used >= time_slice) {
        if (current_queue < 2) {
            new_queue = current_queue + 1;}}

    printf("Process %d moved from Q%d to Q%d\n", process_id,
    current_queue, new_queue);}
int main() {
    update_process_queue(101, 0, 10, 10);
    update_process_queue(102, 0, 10, 3);
    update_process_queue(103, 2, 80, 80);
    return 0;}
```

Output:
Process 101 moved from Q0 to Q1
Process 102 moved from Q0 to Q0
Process 103 moved from Q2 to Q2

(b) (3 points) In lottery scheduling, the scheduler picks a random "winning ticket" to select the next process to run. The C code below simulates finding the owner of a winning ticket. What is its output?

```
#include <stdio.h>
typedef struct {
    char* name;
    int tickets; // Number of tickets this process holds
} Process;
// Finds which process owns the winning_ticket
const char* find_winner(Process procs[], int count,
int winning_ticket) {
    int current_ticket_sum = 0;
    for (int i = 0; i < count; i++) {
        current_ticket_sum += procs[i].tickets;
        if (winning_ticket < current_ticket_sum) {
            return procs[i].name; } }
    return "None"; }
int main() {
```

```
Process process_list[] = {
    {"Process A", 50},
    {"Process B", 25},
    {"Process C", 25}
};
int total_processes = 3;
// Assume the scheduler's random number generator picked 65.
int winning_draw = 65;
printf("The winner is: %s\n", find_winner(process_list,
total_processes, winning_draw));
return 0; }
```

Output:
The winner is: Process B

3. (6 points) (a) (3 points) A process P has been allocated memory given from address 0 to MAX, and it is assigned a memory layout as follows: 10% for code data, 10% for heap, and 10% for stack. In terms of MAX, write the layout of P's view of memory in the system. What are the heap and stack used for?

Ans. P's view of memory would look as follows:
0 to 0.1(MAX) for Static Code
0.1(MAX) to 0.2(MAX) for Heap
0.2(MAX) to 0.9(MAX) will be free
0.9(MAX) to MAX for Stack.

Heap: Stores dynamically allocated data (e.g., via malloc, new). Stack: Stores function call information (return addresses, local variables, parameters).

Scoring Guide: 2 marks for the correct layout, 1 mark for the purpose of stack and heap.

(b) (3 points) Two processes P1 and P2 are running. The following tables describe the base and bounds registers for both:

P1 Segment	Base	Bounds
Code	2000	500
Heap	3000	400

P2 Segment	Base	Bounds
Code	6000	700
Heap	7000	300

Given the following execution trace (assume correct context switches), convert the logical addresses to physical addresses, and also indicate which mode of dynamic relocation is in use. Mention if an error occurs and why?

1. P1 (Code, 200)
2. P1 (Heap, 350)

3. Context switch to P2
4. P2 (Code, 750)
5. P2 (Heap, 150)

Ans. *In User Mode*
P1 Code: $200 < 500$ -> PA $= 2000 + 200 = 2200$
P1 Heap: $350 < 400$ -> PA $= 3000 + 350 = 3350$
In Privileged Mode
Context Switch to P2
In User Mode
P2 Code: $750 \geq 700$ -> Trap/Error
P2 Heap: Would not execute, because process is killed or interrupted at previous step.

An error or trap is raised by the MMU when the logical address for P2 code is greater than the given bounds. In this case, P2 would terminate and no longer execute.

Scoring guide: 2 marks for the correct address translation, and 1 mark for the point at which error occurs and why.

4. (10 points) (a) (5 points) Assume a system with:
Virtual address space $= 2^{32}$ bytes
Physical memory $= 2^{30}$ bytes
Page size = 4KB

Answer the following in one word/number:

1. The number of virtual pages per process? Ans. Number of virtual pages = (virtual address space) ÷ (page size). Therefore, $\frac{2^{32}}{2^{12}} = 2^{32-12} = 2^{20}$
2. The number of physical frames in memory? Ans. Number of physical frames = (physical memory size) ÷ (page size). Therefore, $\frac{2^{30}}{2^{12}} = 2^{30-12} = 2^{18}$
3. The number of bits in the page offset? Ans. 32-20 = 12
4. The number of bits in the virtual page number (VPN)? Ans. 20
5. The number of bits in the physical frame number (PFN)? Ans. 18

(b) (5 points) 1. Paging eliminates external fragmentation completely. True or False? Explain. Ans. True. External fragmentation happens when free memory is broken into *variable-sized holes*, too small to be useful (common in segmentation). Paging divides memory into *fixed-size* blocks (pages and frames). Any free frame can be used for any page. Thus, there's no wasted space between allocations.

2. The hardware cache that stores recent virtual-to-physical translations is called? What happens if an address is not found in this cache? Ans. Translation Lookaside Buffer (TLB). If an address is not found in the TLB, it results in a TLB miss, and the MMU must walk the page table.

3. Every entry in a page table must store the entire virtual page number. True or False? Explain. Ans. False. A Page Table Entry (PTE) does not store the virtual page number. The virtual page number is the index into the page table. Each PTE stores a frame number (physical location of the page) and flags (present, dirty, accessed, R/W, etc.).

4. The bit in a page table entry that indicates whether the page has been modified is called? And the bit that indicates if the page has been accessed is called? Ans. The dirty bit (D) indicates whether the page has been modified after it was brought into memory. The accessed bit (A) indicates whether or not a page has been accessed.
5. This 16-bit virtual address has 8 bits for offset: 0000 0011 0101 1010
   Given that VPN 3 maps to PFN 7, calculate the corresponding physical address if the page size is 2^8. Ans. Here the VPN is 0000 0011 or in Decimal $= 3$. The offset is 0101 1010 or in decimal $= 2^1 + 2^3 + 2^4 + 2^6 = 90$. Then the corresponding physical address $= 7(256) + 90 = 1882$.