Quiz #3

May 22, 2025

Section: 1 or 2 **Student ID:** **Name:**

Q1. What are the pros and cons of demand paging? You should write at least one advantage and at least one disadvantage. (1point)

*** Pros (Advantage)**
- **Efficient memory utilization: only the necessary pages are loaded, reduces memory waste due to unused code or data**
- **Faster program start: Processes start executing more quickly because not all pages need to be loaded at launch.**
- **Improved multiprogramming since less memory used per process**
- **Supports virtual memory**

*** Cons (Disadvantage)**
- **Page Fault Overhead: When a page is not in memory, a page fault occurs, which incurs a high cost to fetch the page from disk.**
- **Increased Complexity: The operating system and hardware must support page tables, valid/invalid bits, and page fault handling routines.**
- **Thrashing Risk: If too many pages are being swapped in and out (due to insufficient memory or poor locality), the system can spend more time paging than executing actual processes.**

Q2. Consider the following page reference string:

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7

Assuming demand paging with three frames, how many page faults would occur for the LRU replacement algorithm? (1point)

9

Q3. How does Linux improve TLB reach and reduce TLB misses in large memory workloads? (1point)

Using Transparent Huge Page (THPhuge page) or Using multiple page sizes

Q4. Which multi-threading model is commonly used in modern operating systems, and why? (1point)

One-to-One model
Since each thread is kernel thread, the OS can run multiple threads in parallel on different CPU cores.

Q5. Complete the *main()* function that performs the task of summing all the numbers from 1 to 100,000,000 (*MAX_NUM*) by dividing the workload equally among 5 threads (*NUM_THREADS*). You may declare variables freely within the *main()* function, but do not use any global variables. (1point)

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>

#define NUM_THREADS 5
#define MAX_NUM 100000000

// Thread function
void *thread_summation(void *arg) {
    int start = ((int *)arg)[0];
    int end = ((int *)arg)[1];
    long long *sum = malloc(sizeof(long long));
    *sum = 0;
    for (int i = start; i <= end; i++) {
        *sum += i;
    }
    return (void *)sum;
}

int main() {
    long long total_sum = 0;

    // Implement from here!
    pthread_t threads[NUM_THREADS];
    int ranges[NUM_THREADS][2];
    int numbers_per_thread = MAX_NUM / NUM_THREADS;

    // Create threads
    for (int i = 0; i < NUM_THREADS; i++) {
        ranges[i][0] = i * numbers_per_thread + 1;
        ranges[i][1] = (i == NUM_THREADS - 1) ? MAX_NUM : (i + 1) * numbers_per_thread;
        pthread_create(&threads[i], NULL, thread_summation, (void *)ranges[i]);
    }

    // Join threads and collect results
    for (int i = 0; i < NUM_THREADS; i++) {
        long long *partial_sum;
        pthread_join(threads[i], (void **)&partial_sum);
        total_sum += *partial_sum;
        free(partial_sum);
    }









    printf("result: %lld\n", total_sum);
    return 0;
}
```

* Result

```
$ gcc thread_largesum.c -o thread_largesum
$ ./thread_largesum
result: 5000000050000000
```