CODE:________________________________

# Operating Systems
# Ph.D. Qualifying Exam

Spring 2022

## 1. PAGING

Write a method(s) to implement Last In First Out (LIFO) page replacement algorithm using your favorite programming language with the assumption that the LIFO can hold up to 100 entries. You may use any language provided data structures. The add method assigns a frame/process/page element to the LIFO. When the LIFO is full, the algorithm should update the LIFO appropriately and execute an **“evict(process, page, frame)”** for the page being evicted. You can assume frames are initially assigned incrementally with a counter. When you evict an entry, you can reuse the frame number from the entry that you are removing. **Evict( )** is an OS primitive and you do not need to implement this function, you only need to implement **add( )**. Each entry on your LIFO can be a record (or object) of the form:

struct LIFO_element {int process, page, frame; struct LIFO_element * next);

add(int process, int page); // takes parameters and adds to LIFO; evicts eligible candidate if full

Is the LIFO page replacement algorithm and effective page replacement algorithm? Support your response.

## 2. Memory Management

An operating system is running on a machine with 4GB of physical memory. The virtual memory is implemented as a pure paging scheme. Assume that each page table entry takes one byte

> What is the size of a single level page table if the the virtual address space is 64-bits, page size is 8K?
> What will be the size of an inverted page table for this system?

## 3. Mutual Exclusion

Many processes in an operating system use a shared counting variable (often called a one-up number). Assume there is a single common, shared variable called COUNT. Write a method **Get_count( )** returns the current value of COUNT and increments COUNT atomically. You are to use OS provided libraries and tools that allow you to guarantee that COUNT is always updated correctly and avoids race conditions.

## 4. OS Performance

Consider a demand-paging system in which processes are performing sequential data accesses with the following time-measured utilizations:

| | |
|---|---|
| CPU utilization | 20% |
| Paging disk | 98% |
| Other I/O devices | 10% |

For each of the following, indicate yes or no and provide a short supporting sentence on whether the proposed change is likely to ***<u>improve CPU utilization</u>***:

a. Install a faster CPU

b. Install a bigger paging disk.

c. Increase the degree of multiprogramming.

d. Decrease the degree of multiprogramming.

e. Install more main memory.

f. Install a faster hard disk.

g. Install multiple controllers with multiple hard disks and stripe the data across the disks.

h. Add prepaging to the page-fetch algorithms.

i. Increase the page size.

j. Increase the I/O bus speed.

## 5. Deadlock Avoidance

What is Deadlock Avoidance? Give an example algorithm that implements deadlock avoidance. What are the requirements for deadlock avoidance?

What is Deadlock Prevention? How is Deadlock prevention implemented? How effective is deadlock prevention?

## 6. Concurrency

```
main() {
      int a = 0;
      int rc = fork();
      a++;
      if (rc == 0) {
           rc = fork();
           a++;
      } else {
          a++;
      }
      printf("Hello!\n");
      printf("a is %d\n", a);
 } // or main
```

7. **Scheduling**

   Suppose there are four processes (P1 - P4) with respective arrival times of 0, 10, 20, and 40, priorities 1, 2, 3, and 4(highest) , and job times of 30, 20, 50, and 20 ms. These processes are scheduled from a single run queue to run on a **dual-processor machine**. Assume the context switch has no overhead.

   What is the average turnaround time for scheduling the four processes using the following preemptive schedulers: priority ( highest-priority-first), round-robin, and shortest time remaining first? For round-robin, assume a 20 ms time quantum.