CS 744 Autumn 2017

Mid-semester Examination

Name	
Roll Number	

Question no.	Marks obtained	Question no.	Marks obtained
1		2	
3		4	
5–6		7–9	
10–19		**Total**	

Please read the following instructions carefully before you start.

- The paper has **19 questions**, for a total of 40 marks, accounting for 25% of your grade.
- You must write your answers in this question booklet itself and turn it in. You need not turn in any rough sheets you may use. Please write **legibly** in the space provided.
- Questions 1–4 carry **5 marks** each and questions 5–9 carry **2 marks** each. You must explain your answers clearly in the space provided.
- Questions 10–19 carry **1 mark** each. For these questions, there is no need to explain your answers.
- Please avoid asking for clarifications, unless you think there is a mistake in the question itself.

1. Consider the readers and writers problem as discussed in class. We wish to implement locking/synchronization between reader and writer threads/processes. We wish to give **preference to writers**, where no waiting writer should be kept waiting for longer than necessary. For example, suppose reader R1 is actively reading. And a writer W1, and then a reader R2, arrive while R1 is reading. While it might be fine to allow R2 in, this could prolong the waiting time of W1 beyond the absolute minimum of waiting until R1 finishes. Therefore, if we want writer preference, R2 should not be allowed before W1. Your goal is to write down pseudocode for read lock, read unlock, write lock, and write unlock functions that the threads should call, in order to realize read/write locks with writer preference. You must use only simple **locks/mutexes** and **conditional variables** in your solution. Please pick sensible names for your variables so that your solution is readable.

   (a) Variables for synchronization and their initial values:

   (b) `ReadLock`

   (c) `ReadUnlock`

   (d) `WriteLock`

   (e) `WriteUnlock`

2. Consider an application that is composed of one master process and multiple worker processes that are forked off the master at the start of application execution. All processes have access to a pool of shared memory pages, and have permissions to read and write from it. This shared memory region (also called the request buffer) is used as follows: the master process receives incoming requests from clients over the network, and writes the requests into the shared request buffer. The worker processes must read the request from the request buffer, process it, and write the response back into the same region of the buffer. Once the response has been generated, the server must reply back to the client. The server and worker processes are single-threaded, and the server uses event-driven I/O to communicate over the network with the clients (you must not make these processes multi threaded). You may assume that the request and the response are of the same size, and multiple such requests or responses can be accommodated in the request buffer. You may also assume that processing every request takes similar amount of CPU time at the worker threads.

   Using this design idea as a starting point, describe the communication and synchronization mechanisms that must be used between the server and worker processes, in order to let the server correctly delegate requests and obtain responses from the worker processes. Your design must ensure that every request placed in the request buffer is processed by one and only one worker thread. You must also ensure that the system is efficient (e.g., no request should be kept waiting if some worker is free) and fair (e.g., all workers share the load almost equally). While you can use any IPC mechanism of your choice, ensure that your system design is practical enough to be implementable in a modern multicore system running an OS like Linux. You need not write any code, and a clear, concise and precise description in English should suffice.

3. Consider a multi-tier server system. The frontend server has several worker threads running on a single CPU core. Upon receiving a request, the server identifies a free worker thread and assigns the request to the worker for all further processing. Processing a request requires 0.1 milliseconds of initial computation in the worker thread, followed by at least 5 milliseconds (may be more if there is queueing) of wait time to perform blocking I/O operations to a backend database. The OS scheduler takes care of context switching out the worker thread during its blocking period, so that CPU cycles are not consumed while waiting for a reply from the backend database. You may ignore the CPU time taken for all other operations, e.g., reading and writing over the network, context switching between threads, and so on.

   (a) Assume that the server uses a fixed pool of worker threads. What is the minimum number of worker threads in the pool that will ensure that the frontend server's CPU is fully utilized? What is the throughput of the frontend server (in units of requests/second) when it is fully saturated?

   (b) Now assume that the server uses a fixed pool of 10 threads to handle the requests. What is the maximum possible throughput of the frontend server?

   (c) From now on, assume that the server does not use a fixed pool of threads. Instead, it spawns a new worker thread every time a request arrives, and the thread exits once the request processing completes. Assume that requests arrive at the frontend server at the rate of 1000 requests/sec, and the average time to process the request and return a response by frontend server is measured to be 10 milliseconds. In this scenario, what is the average number of worker threads in the system at any point of time?

   (d) In the previous part, compute the throughput and utilization of the frontend server.

   (e) Consider the scenario in part (c) once again. Assume that the database has been upgraded to reduce the blocking wait time of a request from 5 milliseconds to 1 millisecond. What metrics (among throughput, utilization, response time) of the server do you expect will change, and how?

4. Consider a computer system with a single core CPU, a single level of cache of size 4MB, and main memory. It takes one CPU cycle to access a memory byte if it is in cache, and 145 cycles if the memory access incurs a cache miss and must be fetched from main memory. The size of the cacheline is 64 bytes. Consider two arrays A and B, each of $N = 2^{20}$ integers (assume that an integer requires 4 bytes of storage). The arrays are stored contiguously in memory, and are aligned at cacheline boundaries. Each case below shows an access pattern of the arrays A and B, and the parameters of the cache. For each case, calculate the average time required to access a *single* element of array A (averaged over all accesses to A in the specified case). Assume that the cache is empty at the start of every scenario, and no other process is using the cache. Assume that the cache does not use any optimizations like prefetching.

   (a) A direct mapped cache, and every element of A is read in sequence as follows.

```
for(i = 0; i < N; i++)
  read A[i];
```

   (b) A direct mapped cache, and every element of A and B is read in sequence as follows.

```
for(i = 0; i < N; i++)
  read A[i];
  read B[i];
```

   (c) A fully associative cache, and the access pattern is as shown in part (b).

   (d) A 2-way set associative cache, and the access pattern is as shown in part (b).

   (e) A 2-way set associative cache, and the access pattern is as follows.

```
for(i = 0; i < N; i++)
  read A[i];
for(i = 0; i < N; i++)
  read A[i];
```

5. Consider a system where each process has a virtual address space of 2^v bytes. The physical address space of the system is 2^p bytes, and the page size is 2^k bytes. The size of each page table entry is 2^e bytes. The system uses hierarchical paging with l levels of page tables, where the page table entries in the last level point to the actual physical pages of the process. Assume $l \geq 2$. Let v_0 denote the number of (most significant) bits of the virtual address that are used as an index into the outermost page table during address translation.

   (a) Derive an expression for l in terms of v, p, k, and e.

   (b) Derive an expression for v_0 in terms of l, v, p, k, and e.

6. Consider a server program running in an online market place firm. The program receives buy and sell orders for one type of commodity from external clients. For every buy or sell request received by the server, the main process spawns a new buy or sell thread. We require that every buy thread waits until a sell thread arrives, and vice versa. A matched pair of buy and sell threads will both return a response to the clients and exit. You may assume that all buy/sell requests are identical to each other, so that any buy thread can be matched with any sell thread. The code executed by the buy thread is shown below (the code of the sell thread would be symmetric). You have to write the synchronization logic that must be run at the start of the execution of the thread to enable it to wait for a matching sell thread to arrive (if none exists already). Once the threads are matched, you may assume that the function `completeBuy()` takes care of the application logic for exchanging information with the matching thread, communicating with the client, and finishing the transaction. You may use any synchronization technique of your choice.

```
//declare any variables here


buy_thread_function:
  //start of sync logic




  //end of sync logic
  completeBuy();
```

7. Consider a 8MB cache with 64 byte cache lines. The system uses 32 bit memory addresses. How many bits are required to be store the tag in a cache entry in each case below?

   (a) The cache is set associative with 8-way associativity.

   (b) The cache is direct mapped.

8. Consider a web server that uses non-blocking event-driven I/O for network communication, but uses blocking I/O to access the disk. The web server wishes to run as multiple processes, so that the server can be available even if some subset of the processes block on disk I/O. Further, the web server wishes to receive web requests only on port 80, and not at different ports in the different processes. Suggest one mechanism by which the multiple server processes can handle requests arriving on a single port on the system.

9. Consider a CPU-bound application server running on a single core CPU. The server receives two types of requests: the harder requests require 10 milliseconds of computation at the server, while the easier requests require 2 millisecond each. Requests arrive at the server at the rate of 20 requests/sec, with the incoming traffic having an equal mix of easy and hard requests. What is the utilization of the server?

10. Consider a process with 9 logical pages, out of which 3 pages are mapped to physical frames. The process accesses one of its 9 pages randomly. What is the probability that the access results in a TLB hit and a subsequent page fault?

11. Consider a process that uses a user level threading library to spawn 10 user level threads. The library maps these 10 threads on to 2 kernel threads. The process is executing on a 8-core system. What is the maximum number of threads of a process that can be executing in parallel?

12. Consider a multicore system where multiple CPU cores have their separate L1 caches, and use the MESI cache coherence protocol between them. A cache line is in the modified state in the L1 cache of core 0, and a thread on core 1 wishes to read the same cache line. What will be the state of the cache line in core 0 after this event?

13. Consider the scenario in the previous question with one change: the thread in core 1 now wishes to write to the same cache line. What will be the state of the cache line in core 0 after this event?

14. A C program is written using the system calls of the POSIX API, and compiled to execute on a machine based on the Intel x86 architecture running a POSIX-compliant version of Linux. We wish to execute this program on a different POSIX-compliant OS running on a system architecture based on a different underlying instruction set. Should the C program have to be rewritten in order to execute correctly in the new environment? Answer yes/no.

15. In the previous question, should the program binary generated for the old environment have to be recompiled to execute correctly in the new environment? Answer yes/no.

16. A process P forks a child process C. P then exits while C is still running. Then, process C becomes a zombie process. Answer true/false.

17. The `mmap` system call can be used to increase the size of the heap of a process. Answer true/false.

18. Consider an OS using demand paging. The page table entry of a page of a process is created only when the page is first accessed (read/written). Answer true/false.

19. A program has an array of N integers, which is shared across N kernel threads. The kernel thread i updates the i-th integer in the array during its execution. This design ensures no sharing of cache lines between the threads when the threads execute on separate CPU cores. Answer true/false.