**OPERATING SYSTEMS WORKSHOP 02**

1. Describe the actions taken by a kernel to context switch between processes.
   1. When a process terminate you can have a context switch.
   2. The kernel must save the context of the old process (stored in the Process Control Block, PCB), load the new processes context, then save the context of that one and load the next.
2. Including the initial parent process, how many processes are created in the following program? Verify your answer by executing the program in a computer.
   1. 8. One becomes two, two becomes four, four becomes eight.
3. Cooperating processes require an inter process communication mechanism to communicate with each other. The two major communication schemes are: shared memory and message passing. Contrast the two mechanisms. What is the responsibility of the OS for each mechanism?
   1. Shared Memory: OS must create shared memory and then attach the shared processes to the memory region address space, also checking that the processes have the right to access the memory.
   2. Message passing: OS first creates the buffer for messages and the handles system calls to access the buffer. Must also handle all send and receive calls.
4. The send() and receive() used in the message-passing system for inter process communication may apply the following methods:
   1. Synchronous and asynchronous communication
      1. Blocking or non-blocking.
      2. Synchronous be good for purposes where a process knows that it is receiving. Asynchronous bad if we don’t know what the processes are doing?
   2. Automatic and explicit buffering
      1. Allocation of specific size of buffer or not.
      2. Explicit can ensure no overflow of buffer. Explicit however can also be bad as it doesn’t change and can overflow while automatic resizes.
   3. Send by copy and send by reference
      1. Message can be stored in memory or sending the actual message.
      2. Send by reference can speed up the sending/processing of a message. Send by copy can be limited as it can’t be modified. For small message send by reference can take longer.
   4. Fixed-sized and variable-sized messages
      1. Messaged have to stay a certain size and be broken up or can be any.
      2. Fixed size can ensure a specified number of messages stored in a fixed buffer. Can be a detriment if the message is too big.
5. Discuss the differences, advantages and disadvantages of the user-level threads and the kernel-supported threads.
   1. User-level thread
      1. Good as you can create as many threads as you want (considering resources) and are faster (more efficient) as they are run by libraries and not the OS.
      2. Not great if the user-level thread needs I/O or something will have to share its OS time with all the user-level threads (slower).
   2. Kernel-level threads
      1. Good as if one thread is blocked another thread of the same process can still run
      2. Not great as they are slower because they are run by the OS.
6. Describe the actions taken by a thread library to context switch between user level threads.
   1. Easier than context switching processes. Only need to save the register set of the old thread and then load the new one.
7. Under what circumstances does a multithreaded solution using multiple kernel threads provide better performance than a single-threaded solution on a single-processor system?
   1. If one of the threads encounters a fault an error will not occur.
8. Consider a multiprocessor system and a multithreaded program written using the many-to- many threading model. Let the number of user-level threads in the program be greater than the number of processors in the system. Discuss the performance implications when the number of kernel threads allocated to the program is less than the number of processors.
   1. It is not performing as well as it can as it isn’t utilising all of the processes as only the amount of kernel-level-threads will be used.