University of Newcastle

School of Electrical Engineering and Computer Science

**COMP2240 - Operating Systems**

**Workshop 2-Solution**

**Topics: Process and Threads**

|  |  |
| --- | --- |
| **1.** | Consider an environment in which there is a one-to-one mapping between user-level threads and kernel-level threads that allows one or more threads within a process to issue blocking system calls while other threads continue to run.  Explain why this model can make multithreaded programs run faster than their single-threaded counterparts on a uniprocessor computer.  **Answer:**  The issue here is that a machine spends a considerable amount of its waking hours waiting for I/O to complete. In a multithreaded program, one KLT (kernel-level threads) can make the blocking system call, while the other KLTs can continue to run. On uniprocessors, a process that would otherwise have to block for all these calls can continue to run its other threads. |
| **2.** | Consider the state transition diagram given in Figure 1:    Figure 1: Process State Transition Diagram with Two Suspend States  Suppose that it is time for the OS to dispatch a process and that there are processes in both the Ready state and the Ready/Suspend state, and that at least one process in the Ready/Suspend state has higher scheduling priority than any of the processes in the Ready state.  Two extreme policies are as follows:  (1) Always dispatch from a process in the Ready state, to minimize swapping  (2) Always give preference to the highest-priority process, even though that may mean swapping when swapping is not necessary.  Suggest an intermediate policy that tries to balance the concerns of priority and performance.  **Answer:**  Penalize the Ready, suspend processes by some fixed amount, such as one or two priority levels, so that a Ready, suspend process is chosen next only if it has a higher priority than the highest-priority Ready process by several levels of priority. |
| **3.** | Figure 1 suggests that a process can only be in one event queue at a time.   1. Is it possible that you would want to allow a process to wait on more than one event at the same time? Provide an example. 2. In that case, how would you modify the queueing structure of the figure to support this new feature?   **Answer:**   1. An application may be processing data received from another process and storing the results on disk. If there is data waiting to be taken from the other process, the application may proceed to get that data and process it. If a previous disk write has completed and there is processed data to write out, the application may proceed to write to disk. There may be a point where the process is waiting both for additional data from the input process and for disk availability. 2. There are several ways that could be handled. A special type of either/or queue could be used. Or the process could be put in two separate queues. In either case, the operating system would have to handle the details of alerting the process to the occurrence of both events, one after the other. |
| **4.** | Consider a multicore 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 processing cores in the system. Discuss the performance implications of the following scenarios.   1. The number of kernel threads allocated to the program is less than the number of processing cores. 2. The number of kernel threads allocated to the program is equal to the number of processing cores. 3. The number of kernel threads allocated to the program is greater than the number of processing cores but less than the number of user-level threads.   **Answer:**  When the number of kernel threads is less than the number of processors, then some of the processors would remain idle since the scheduler maps only kernel threads to processors and not user-level threads to processors.  When the number of kernel threads is exactly equal to the number of processors, then it is possible that all of the processors might be utilized simultaneously. However, when a kernel thread blocks inside the kernel (due to a page fault or while invoking system calls), the corresponding processor would remain idle.  When there are more kernel threads than processors, a blocked kernel thread could be swapped out in favor of another kernel thread that is ready to execute, thereby increasing the utilization of the multiprocessor system. |
| **5.** | In the given text, it discussed Google’s Chrome browser and its practice of opening each new website in a separate process.    Would the same benefits have been achieved if instead Chrome had been designed to open each new website in a separate thread? Explain.  **Answer:**  No because the whole reason why a new process was created for each browser was because if a webpage crashed, it wouldn't crash the whole browser. If you use a thread for each webpage, then if a webpage crashes, it will bring down the whole application. |
|  |  |

**Supplementary problems:**

|  |  |
| --- | --- |
| **S1.** | What is a zombie process? What is the importance of a zombie process?  **Answer:**  A process that executes an exit system call is first placed in the zombie state. A process in the zombie state does not exist anymore. It may be leaving (timing) information for its parent and an exit code specifying the reason for exiting. The zombie process disappears when a wait system call is executed by the parent process. The exit call does the following:  Release all memory used by the process except the user structure  Reduce reference counts for all open files, and close if necessary  Release any shared text area  Mark process state as zombie  Copy exit status into process table entry and release user structure  Search process table for parent process and child processes  If child is in zombie state, signal init process  Else, have child processes adopted by init process  Switch to another process |
| **S2.** | A user's text segment is re-entrant and can therefore be shared. Data and stack segments, on the other hand, are private. What does this mean? Why is it significant?  **Answer:**  If two or more users access the same program, only one text segment is physically stored in memory. Both users will have independent process images (text, data and stack segments), but physically they share a single text segment. The private data and stack segments allow the user processes to execute the same code, but not interfere with each others input and output data. Their execution is totally independent.  Each process will also have its own system data segment containing information needed by the operating system when the process is active. This system data segment is not part of the user's image, the user can not access it. When the user calls the system (e.g. to request I/O), the process switches from user state to system state, making the system data segment available to the kernel |
| **S3.** | Which type of process is generally favoured by a multi-level feedback queuing scheduler: a processor-bound process or an I/O-bound process? Briefly explain why?  **Answer:**  If a process uses too much processor time it will be moved to a lower-priority queue. This leaves I/O-bound processes in the higher-priority queue. UNIX System V (pages 422-423) is an example of this scheme. |
| **S4.** | A system with two dual-core processors has four processors available for scheduling. A CPU-intensive application is running on this system.  All input is performed at program start-up, when a single file must be opened. Similarly, all output is performed just before the program terminates, when the program results must be written to a single file.  Between startup and termination, the program is entirely CPUbound. Your task is to improve the performance of this application by multithreading it. The application runs on a system that uses the one-to-one threading model (each user thread maps to a kernel thread).   1. How many threads will you create to perform the input and output? Explain. 2. How many threads will you create for the CPU-intensive portion of the application? Explain.   **Answer:**   1. It only makes sense to create as many threads as there are blocking system calls, as the threads will be spent blocking. Creating additional threads provides no benefit. Thus, it makes sense to create a single thread for input and a single thread for output. 2. Four. There should be as many threads as there are processing cores. Fewer would be a waste of processing resources, and any number > 4 would be unable to run. |