Benjamin Barnes
CIS415
Assignment 1
Joe Sventek

1. **What are the two models of interprocess communication? What are the strengths and weaknesses of the two approaches?**
   The two models are the message-passing model and the shared-memory model. An advantage of message passing is that it is easier to implement than shared-memory and usually works better with small amounts of data as it results in fewer conflicts. A disadvantage of message passing is that there is a possibility that the computers are restricted by the speed of the (potential) network passing the messages. An advantage of shared-memory is that it is much faster than message passing as the messages are passed at memory speeds. A disadvantage of memory passing is that now one has to worry about protection and synchronization of data.
2. **Why is the separation of mechanism and policy desirable?**
   Without the separation of mechanism and policy, even small changes to the policy would require the mechanism to change. With them being separated appropriately, they are independent of each other which allows a flexible design.
3. **Describe the actions taken by a kernel to context-switch between processes?**
   When doing a context-switch, the processor must save the state of the current process and then load in the saved context of the process it is about to execute. When the state of the current process is being saved, the kernel is concerned with keeping the registers, memory management, and process state.
4. **What are the benefits and the disadvantages of each of the following? Consider both the system level and the programmer level.**
   a. **Synchronous and asynchronous communication**
      The benefit of synchronous communication is that it allows for a rendezvous between the two – essentially setting us up for a producer and consumer relationship. Asynchronous communication allows for the processes to continue which can result in processes finishing quicker but does not offer the security that synchronous communication does.
   b. **Automatic and explicit buffering**
      A benefit of automatic buffering is that the sender doesn't ever have to worry about putting too many messages in the buffer as it automatically resizes when more space is needed. The obvious potential disadvantage to this is that the buffer could take up an incredible amount of space if the receiver is not processing the messages. An advantage of the explicit buffer is that we don't have to worry about it taking up too much memory as the size is restricted but this can end up making the sender block if the buffer is full.
   c. **Send by copy and send by reference**
      Sending by copy does exactly what it sounds like – sends a copy of the data to the receiver. Any changes they make to it do not change the original item. This can be more demanding, especially if the data set is large as we literally pass a copy. Passing by reference allows the data set

to be changed as we can dereference the reference to access it. Because it is just a reference, this is usually less intensive than passing a copy.

   d. **Fixed-sizes and variable-sized messages**
   Through using fixed-size messages, we make implementation on a system level easy, but in doing so we make it more difficult on the programmer level. The opposite is true for variable-sized messages: it becomes harder to implement on the system level but easier on a programmer level.

5. **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 start-up and termination, the program is entirely CPU-bound. 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).**
   a. **How many threads will you create to perform the input and output? Explain.**
   I would use 1 thread as we most likely need to receive the input in in-order and producing multiple threads to read in from could potentially lead to collisions (reading something in twice, etc). We would need to have a lock on the data when a thread was reading to prevent this, so adding multiple threads would just result in them operating sequentially. In adding more threads, we are also introducing more overhead than necessary, so our program could potentially run slower with more than 1 thread. The same logic applies to the output.
   b. **How many threads will you create for the CPU-intensive portion of the application? Explain.**
   Because the time the program is not starting up or shutting down, it is CPU-bound - we are able to make use of all four cores. I would create 4 threads so that each one has its own core to work with, as adding more than that most likely results in unnecessary overhead.
6. **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.**
   a. **The number of kernel threads allocated to the program is less than the number of processing cores.**
   In this situation, we would have cores that are doing nothing during the computation because the kernel is what assigns the threads to cores.
   b. **The number of kernel threads allocated to the program is equal to the number of processing cores.**
   In this situation all cores would be used during the computation as the kernel assigns a thread to each of the cores.

c. **The number of kernel threads allocated to the program is greater than the number of processing cores but is still less than the number of user-level threads.**
   Because we have more threads than necessary, we could potentially lower performance as we are constantly switching between threads (no work is done when switching) when in reality we could have had better performance with # of threads = # of cores.

Process Analysis

When designing a methodology the get information about a process, the first area I would go look at is the status file of the PID. The status file is a good start as it contains a majority of the information in '/proc/PID/stat' and '/proc/PID/statm' except that it is formatted to be easily read by humans. Upon executing 'cat status' while in the PID subdirectory, one is able to find out the overall state of the process. In 'status' is the process name, its current state, as well as all the statistics on its virtual memory usage. Specially, we would look at the current state to see if the process is in a Zombie state and kill it if we find it necessary. Also, we could look at 'VmPeak' to see the peak memory usage as well as 'VmData' to see the size of the processes data segment. These two Vm status lines could be particularly important if they are abnormally large - they could be slowing our operating system down and might be worth killing (obviously depending on their importance). One thing that would also be worth gathering is the number of threads running for this process. This also lies in the status file under 'Threads'

Another important file I would check is '/proc/PID/cmdline' as it shows us how the process is executed via the command line. In exploring this file we can figure out how to make future calls to the process.

Lastly, I would check the '/proc/interrupts' file to see what files are causing interrupts. Observing this file, one could see if the process they are interested in is causing any specific interrupts.