Keith Schmitt

CIS 452:10

Professor Dulimarta

1/30/2018

**4.2 What are two differences between user-level threads and kernel-level threads? Give an example of an application for which one type might be better than the other?**

Kernel Level Threads: They are threads of execution of a process that run concurrently. These types of threads are managed by the operating system and are relatively lightweight compared to entirely new processes. (1 difference) They are close to the kernel and are managed by the operating system and can view other threads in the kernel. (2 difference) The use system calls to manage the threads of a process and are visible to the Operating System. Kernel Level Threads are also relatively slower compared to User Level Threads.

User Level Threads: They are also used as threads of a concurrent execution of process.

(1 difference) They are also completely managed by the user level library. (2 difference) These types of threads are interpreted as a single threaded process by the Operating System. These types of threads can also be used on a system that does not inherently support threading.

You may want to use a kernel level thread when you are using a multiple of block statements, where the operating system can better manage the processes rather than simulated threading

You may want to use user level threads where the type of system being used does not support threading inherently. Or if you want to avoid the cost of switching between kernel threads.

**4.8 Which of the following components of program state are shared across threads in a multithreaded process?**

a. Register values –NOT SHARED

**b. Heap memory -SHARED**

**c. Global variables -SHARED**

d. Stack memory –NOT SHARED

Practical: The file /kernel/kthread.c contains a function called kthread\_bind().

• What does this function do?

This function binds a thread to a cpu.

• Specifically, why might this function be called?

If you have a multiple core system and want to assign a thread in order to optimize or perhaps trace what is being prioritized in the execution of a multithreaded process.

o Hint: think of a multi-core CPU and how this function might benefit thread performance.