Ryan Morris

[31 out of 40]

os ch 4 test

1. [3] What is the difference between multi-processing and multi-threading? **Processes, spawned from a single program/process each have there own unique stack and run independently of each other on their own,** **registers and code space in memory. Threads share the same stack, and memories registers this allows them to share resources and pass information much more easily then processes. Both operate concurrently (when possible) but thread execution is governed by the operating system, or thread library.**

2. [1] What is typically shared between threads of a single process? What is not? **Registers, stacks, and memory are shared between typical threads of single process, however code is not. Also Linux makes no designation between threads and processes and calls them tasks instead. Tasks started by the clone() command take flags as an argument that determine what resources are to be shared between 2 running tasks. So potentially File system information, memory space, signal handlers or files are shared or not shared.**

3. [2] How do you start a new thread in some API? (pick one) pthread\_create(&jobs[i].tid,&jobs[i].attr,worker,&jobs[i]);

4. [4] What are some advantages to multi-threaded programming. **Multi threaded programming allows certain time computationally expensive tasks to be parallelized to speed up execution time, it also for programs dependent on blocking actions to be made more responsive. For instance servers waiting for connections spawn the new connection on a new thread so they can continue waiting for connections. Gui's can be be run on a different thread then the main program that way they are always responsive. Also threads share the same resources so it is often more economical to use a multi-threaded approach then a multi-process approach.**

5. [4] What are difficulties with multi-threaded programming compared to single-threaded application development. **Certain tasks don't lend well to parallelization and hence multi-threading I/O intensive tasks and tasks that depend on the result of a previous action can not be effectively multi-threaded. Resource usage can be an issue since all threads run in a shared memory space and use the same resources. One thread hogging all the ram will slow down or stop other threads that depend on the same memory space.**

6. [4] When would multi-processing be a better choice than a multi-threaded approach? **When two or more tasks that are resource intensive need their own (non-shared) resources to accomplish a task. Or when you have multiple tasks and one of them is very important, and you don't want to potential of a thread executing in a shared memory space to crash or corrupt the other thread.**

7. [4] What is the idea of processor/core affinity? **Processor/core affinity is the the approach of restricting a process/thread to a single core/processor in order to avoid the high cost of invalidating and repopulating caches. It can be enforced by the operating system in soft (the operating system attempts to keep the process running on the same processor) and hard (the operating system forces the processes to run on the same processor) affinity.**

8. [4] What is the purpose of the argument to the thread entry function, in pthreads, written like...

void\* worker(void \*arg)

{

}

pthread\_create(...,worker,...);

**Worker is a function that executes arbitrary code on a separate thread. Pthread\_create(... worker …) is the function that creates a new Posix thread and tells it to execute the function worker.**

9. [3] How are number of threads and number of cores related? **The number of threads that can concurrently execute is equal to the number of (logical cpus) cores in the system and so the operating system will often run threads on their own cores so they can have full cpu time and the operating system doesn't have to context switch and time slice the execution of threads. However thread scheduling is controlled by the operating system, so there is no guarantee any one thread will run on any one core.**

10. [2] What are some considerations about stack space and new threads? **New threads execute in the same stack space as their parent thread and other threads spawned by the same process. As such if there are too many other threads or if one thread is using too many resources the stack can be filled up and the program (and all threads associated with it) will hang or crash.**