CSE 421/521 – Operating Systems
Fall 2014 - Homework Assignment #1

**** SOLUTIONS *****

Problem 1:

Direct memory access is used for high-speed I/O devices in order to avoid increasing the CPÚs execution load.

a. How does the CPU interface with the device to coordinate the transfer?

b. How does the CPU know when the memory operations are complete?

c. The CPU is allowed to execute other programs while the DMA controller is transferring data. Does this process interfere with the execution of the user programs? If so, describe what forms of interference are caused.

Answer:

a) The CPU can initiate a DMA operation by writing values into special registers that can be independently accessed by the device. The device initiates the corresponding operation once it receives a command from the CPU.

b) When the device is finished with its operation, it interrupts the CPU to indicate the completion of the operation.

c) Both the device and the CPU can be accessing memory simultaneously. The memory controller provides access to the memory bus in a fair manner to these two entities. A CPU might therefore be unable to issue memory operations at peak speeds since it has to compete with the device in order to obtain access to the memory bus.

Problem 2:

What is the main advantage of the microkernel approach to system design? How do user programs and system services interact in a microkernel architecture? What are the disadvantages of using the microkernel approach?

Answer:

Benefits typically include the following (a) adding a new service does not require modifying the kernel, (b) it is more secure as more operations are done in user mode than in kernel mode, and (c) a simpler kernel design and functionality typically results in a more reliable operating system.
User programs and system services interact in a microkernel architecture by using interprocess communication mechanisms such as messaging. These

messages are conveyed by the operating system.
The primary disadvantage of the microkernel architecture are the overheads associated with interprocess communication and the frequent use of the operating system's messaging functions in order to enable the user process and the system service to interact with each other.

Problem 3:

Give detailed answer to each of the following questions:

(a) What is the difference between fork() and exec() on Unix?

(b) What resources are used when a thread is created? How do these differ from those used when a process is created?

(c) What are context switches used for and what does a typical context switch involve?

Answer:

a) **fork()** creates a new process and copies the address space from the parent into the new child process. **exec()** stops executing the program in the process, overwrites it with a new program, and starts executing the program at the beginning; exec() does not create a new process.

b) Threads are created *within and belonging to* processes. All the threads created within one process share the resources of the process including the address space. On the other hand, each **process** is created with their own resources, such as a separate address space.

c) When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process, which is called context-switch. During a context-switch, the kernel saves the context of the old process in its PCB, stops the old process, loads the context of the new process from its PCB, and starts the new process.

PROBLEM 4:

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 more than the number of processors 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 processors.

b. The number of kernel threads allocated to the program is equal to the number of processors.

c. The number of kernel threads allocated to the program is greater than the number of processors but less than the number of user-level threads.

Answer:

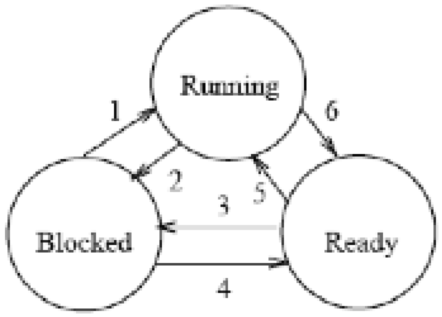
a) When the number of kernel threads is less than the number of processors, then some of the processors would remain idle since the scheduler maps only kernel threads to processors and not user-level threads to processors.

b) 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.

c) 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.

Problem 5:

As shown below, processes can be in one of three states: running, ready and blocked. There are six possible state transitions (labeled 1-6). For each label, indicate whether the transition is valid or not valid. If valid, indicate when the transition is used for a process (i.e. give an example). If the transition is not valid then indicate why.



Answer:

State transitions:

(a) 1: Blocked to Running

Not valid. Must go to ready state before running.

(b) 2: Running to Blocked

Valid. Process waits for I/O or another event to happen.

(c) 3: Ready to Blocked

Not valid. No reason to be blocked while in the ready queue.

(d) 4: Blocked to Ready

Valid. I/O or event completion occurs.

(e) 5: Ready to Running

Valid. CPU becomes available and next process in the ready queue gets dispatched.

(f) 6: Running to Ready

Valid. An interrupt occurs and the running process is preempted.