CSE 421/521 – Operating Systems **Sassi Kesavan**
Fall 2018 - Homework Assignment #1 **UBIT: sassikes**

Problem 1:

What is the main advantage of 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?

- *Advantages:*
  - *Smaller kernel because of removing nonessential components from the kernel and implementing them as the system or user programs.*
  - *Extending the operating system is easier. New services are added to user space and no kernel modification is required.*
  - *Provides more security and reliability as most services run as user processes.*
- *Disadvantages:*
  - *The performance of the microkernel can suffer due to increased system function overhead.*
- *The user program and system services interact with each other through message passing.*

Problem 2:

a) What is the difference between fork() and exec() system calls in Unix?
   - *Fork() system call creates a new process which consist of copy of address space of the original process. Both the process will continue to execute from the next instruction after the fork() call. Return code for the child process is zero where as the return code of the parent process is the process id of the child process which is nonzero. If fork() system call fails, it will return -1.*
   - *Exec() system call replaces current process image with the new image. The text, data and stack of the process are replaced.*

b) What resources are used when the thread is created? How do these differ from those used when process is created?
   - *When a process is created, it will need certain resources such as CPU time, memory, files and IO devices to accomplish its task. Since thread is smaller than process it only requires fewer resources to create.*

c) What are context switches used for and what does typical context switch involve?
   - *Context switching is a procedure that the CPU follows to change from one process to another.*
   - *During the context switch, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run. The system needs to save the current context of the process running on the CPU so that it can restore that context when its processing is done, essentially suspending the process and then resuming it. The context is represented in the PCB of the process.*

Sassi Kesavan
UBIT name: sassikes

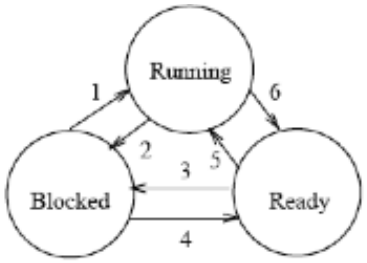
Problem 3:

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 greater 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.
   If the number of kernel thread is less that the number of processor then few processors may remain idle.
b) The number of kernel threads allocated to the program is equal to the number of processors.
   If the number of kernel thread is equal to the number of processors, then all the processors are utilized. But, if any thread enters the wait state then that processor will be idle.
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.
   If the number of kernel threads are greater than the number of processors, all the processors will the utilized all the time. When compared to the previous scenario the utilization of the processors is higher because when a thread waits it is swapped with the new thread that is ready for execution.

Problem 4:

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.



State transitions:

(a) 1: Blocked to Running- *Not valid. The process in the block state cannot go back to the running state directly. The scheduler will not place the process in the block state. The process in the blocked state can be executed only if it comes to the ready state.*

(b) 2: Running to Blocked- *Valid. When a running process requests for an I/O or system call, it is placed in the blocked state.*

(c) 3: Ready to Blocked- *Not valid. The process in the ready queue cannot be put in the block state directly.*

(d) 4: Blocked to Ready: *Valid- When the process in the blocked state completes the I/O operation or a system call, it needs to be put in the ready state to get the CPU for execution.*

(e) 5: Ready to Running: *Valid- The new process that are placed in the ready queue to get the CPU for execution. The scheduler will schedule the process to run in the CPU.*

(f) 6: Running to Ready: *Valid- Only if a high priority process comes to the ready queue state or if the timer set for the process in the CPU expires.*

Problem 5:

In the code below, assume that

```
void main()
{
        pid1 = fork();
        if (pid1 == 0) {
                pid2 = fork();
                printf("A");
        }
        else{
                execvp(…)
        }
        printf("B");
        pid3 = fork();
        if (pid4 != 0) {
                printf("C");
                execvp(…);
        }
        else{
                if (pid1 != 0){
                        pid5 = fork();
                        execvp(…);
                        printf("D");
                }
        }
        if (pid2>0){
                pid6 = fork();
                printf("E");
        }
        else{
                printf("F")
                execvp(…);
        }
        printf("G");
}
```

(i) all fork and execvp statements execute successfully

(ii) the program arguments of execvp do not spawn more processes or print out more characters

(iii) all pid variables (pid,…,pid6) are initialized to 0.

(a) How many processes will be created by the execution of this code? Show in a "process creation diagram" (like we did in the class) the order in which each process is created, and the values of pid 1 to pid6 for each process.

(b) What will be the output of each process?

Sassi Kesavan
UBIT name: sassikes