|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***COMP3300*** |  |  | ***Homework # 2*** |  |  | **Total Points: 100** |

**Handed Out: Thursday Jan. 24, 2019    Due: Tuesday Feb. 05, 2019**

**Important:**

-        This is an individual homework assignment. You are not allowed to collaborate with other individuals or Internet to do the work.

-        Questions 1 – 4: 15 points each; Questions 5 & 6: 20 points each.

-        Please type your answers.

-        As per policy stated in the course outline, homework is due at the beginning of class time on its due date. All submissions will be through the Blackboard site only. Late submissions will be subjected to policy indicated in the course outline.

-        For this homework assignment, **ONLY one submission** attempt will be allowed.

**Name: Kolby Sarson**

**ID: 104232327**

**Question 1**: Compare and contrast the following types of operating systems. Be sure to include when they are or are not appropriate to use.

-        Batch

o    The user and computer do not directly interact. Jobs with similar needs are grouped into batches and the system runs each program, or batch, one at a time. They are best used when independent processing is needed (users are not needed for processing). A payroll system is one instance of where this would be useful since payroll should automatically be made available to employees at the end of the pay period

-        Interactive

o    Similar to batch except there is more interaction between users and the computer. Windows and Mac are two examples of interactive OS. They are appropriate when user interaction is a must.

-        Time-slicing (time-sharing)

o    In a Time-Sharing OS, each user is allotted a portion of the resources and the system switches between all users rapidly for seconds at a time. This process is quick enough that users are given the impression that they are the only user. This reduces CPU idle time. Examples include Unix and Multics. This is an appropriate system to use when multiple users require access to the system simultaneously.

-        Real-time

o    In a real time OS, the system quickly switches between tasks, running only one program at a time using Threads. This OS is appropriate when small response times are crucial, such as missile systems and air traffic control systems.

-        Distributed

o    A Distributed System helps manage resources when multiple machines are working cooperatively to carry out computations. An online examination system is one such example, since multiple users are accessing a single distributed system to view a file containing the examination questions.

**Question 2**: What are the differences between a program, an executable, and a process?

* Program
  + A set of instructions which is in human readable format (HelloWorld.c)
* Executable
  + A compiled form of a *Program*, which can be acquired through the use of a compiler such as Cygwin (HelloWorld.exe file)
* Process
  + The executable when it is being run by the OS. This is what you see in the Task Manager (HelloWord.exe Process when we double click it)

**Question 3**: Let an OS supports a system call *sleep*, which puts the program making the call to sleep for the number of seconds indicated in the argument of the *sleep* call. Explain how this system call may be implemented. [**HINT:** think in terms of timer]

Sleep() could be used as a form of limiting both the hardware and software of a system to avoid overheating of the hardware and maintain proper timing of cooperating processes.  Sleeping for n seconds where n is enough time for the hardware to cool enough to continue working or enough time for processes’ executions to be timed properly.

In terms of timers, sleep() could be implemented alongside a timer, or could be used as a timer itself. In the former, a timer queue is checked after each scheduler tick. If a given timer has expired, which indicates that it is time to wake the corresponding process, then that process is awoken. In the latter, we use sleep() on its own, waking a process periodically to check if it’s time to run, if not we return the process to a sleeping state.

**Question 4**:  What happens on a context switch? Should context switches happen frequently or infrequently? Explain your answer.

During a context switch, the OS saves the current execution context in the OS’s PCB for this process and changes the program to the appropriate status (ready, waiting, or terminated).  Then the next process in the ready queue is selected for execution, the status is changed to “running”, and its execution context is loaded from the OS’s PCB for this process onto the hardware.  A context switch should happen frequently enough so that all jobs in the system make process, but infrequently enough that their overhead does not take up a significant amount of the total system CPU time.  Usually, around 1 to 2% of total system time is an appropriate context switch time.

**Question 5**: Consider an interprocess-communication that uses mailboxes:

mailbox:send(ToID, msg)    // send msg to Process Id, ToID mailbox:receive(FromID, msg) // receive msg from process id, FromID

Is it possible for a process to wait to get a message from any one of a number of processes? If your answer is yes, write a program pseudo code that does so.

Yes, it is possible for a process to wait to get a message from any one of a number of processes, assuming that the processes have a shared mailbox.  For this problem, assume that “any one of a number of processes” is non-specific, waiting for a message from any of N processes, regardless of which process sends the message

main(){

Create parent process

Create N child processes

//child processes

if(Conditional for sending a message)

send(ToID, msg)

die(Pid)

// parent process

wait(&childPid)

receive(FromId, msg) //FromID = childPid

}

Now, consider the case that we add a new mailbox primitive:

mailbox:empty(FromID)  // returns false if no msg in mailbox,

// return true otherwise

Use it to write pseudo code for a program that waits for a message from any one of a number of processes. Assume there are total of **n** processes, numbered: **0** through **(n-1)**.

// Assume we have an array of Pid of size n

main(){

Create parent process

Create N child processes

//child processes

if(Conditional for sending a message)

send(ToID, msg)

// parent process

for(process i, 0 to n-1)

If (!empty(Pid i))

receive(FromId, msg) //FromID = Pid of i

}

**Question 6**: 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

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.

In this case, some of the processors would remain idle since there are not enough kernel threads to accommodate the processors (scheduler maps only kernel threads to processors).

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

In this case, all the processors would be utilized simultaneously since there are enough kernel threads to accommodate the processors.  However, a process may idle if a thread blocks inside the kernel.

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.

In this case, since we have an excess of kernel threads, a blocked kernel thread could be swapped out if necessary, increasing the utilization of the multiprocessor system.