CS3510-Operating systems

Final Exam

MS17BTECH11013

P. Pavan Kalyan

1.

JOB1 total time – 23ms, CPU time – 3ms.

JOB2 total time – 29ms, CPU time – 5ms

JOB3 total time – 14ms, CPU time – 4ms.

CPU utilization = totalTime / CPU exe Time

CPU utilization for job1 is 23/3 = 7.66

CPU utilization for job2 is 29/4 = 7.25

CPU utilization for job3 is 14/4 = 3.5

For uniprogramming

The CPU will execute other process only when it finishes the current process.

So the CPU utilization for all jobs will same as calculated before.

First job1 is executed, it will execute for a time of 23ms. This job involves I/O activity, CPU time and another I/O activity. Even when the process goes for I/O the cpu remains idle and when the I/O finishes process continues execution for 3ms and then again goes for I/O operation.

For Multi Programming

Some process will run at all times so as to maximize CPU utilization. We can see that among the 3 jobs the maximum CPU utilization is for job1 and then job2 then job3.

5. Given an application has 20% of code as serial.

(Assume that there is no other else serial code and only parallel is present).

Then the total series code is 20% and parallel code is 80%. This I assume to be the theoretically max condition.

We know from Amdal's law that

Speedup <= 1 / (S + (1-S)/N)

Where S = series portion

N = processing cores

a) N = 4 (4 processors)

The theoretical maximum speed up is

1 / (0.2 + 0.8/4) = 2.5

b) N = 8 (8 Processors)

The theoretical maximum speed up is

1 / (0.2 + 0.8/8) = 3.33

6. The application where thread pools are very useful is multithreaded web server. Because a server might get multiple requests from the clients and creating a separate process for it really time consuming and huge whereas thread creation is light weight but creating a new thread of all requests from clients is sometimes huge. So to limit the number of threads, we can thread pool. Where a certain number of threads are created at the start-up and are present in a pool and wait for a request/work. So rather than creating a thread the server submit the request to the thread pool. If there is any thread which is available in the pool will be assigned the request. If there are no threads then the task is queued until a thread finishes its job.

7. No,

Given that there are several independent calculations which are running concurrently. So as tasks are independent, we can make use of threads efficiently to run them parallel to decrease the total time. Concurrent execution means time sharing means at any point of time we can run only one task but having run them parallel with multiple threads which are working on independent calculation we can have more tasks running at any given instant of time. Which can decrease the time.

So, there is no circumstance in which an algorithm that performs several independent calculations concurrently be more efficient if it is did not use threads.

3. Given 64-bit microprocessor and 64-bit instructions.

Given that it has opcode occupies the first 4 bytes (4*8 = 32 bits) and the rest (64-32 = 32 bits) is for immediate operand/Operand address. Then the maximum directly addressable memory capacity is around (2^32 – 1) bits. (Assuming that the minimum memory capacity is starting from 0)

2. Given that teletype is a simple keyboard (encode a symbol) / printer (decode a word).

Keystroke input from the teletype and output to the printer are controlled by the I/O module.

The I/O module had 5 registers INPR, OUTR, FGI, FGO, IEN

a) Given the job is to achieve I/O with the teletype. We need to use the first 4 registers only.

For input the input flag is set now the I/O module reads the data which the teletype has encoded to an 8-bit word. It reads that and writes it in the input register. The CPU reads the data and after finishes it job it unset the input flag again.

For output the output flag is set now the I/O module reads the data from the output register which the CPU has written to output to the printer. I/O module reads it and sends it to the teletype. The teletype decodes an 8-bit word into an alphanumeric symbol and prints it.

b) When we employ IEN flag. It set the IEN flag when a process wants to do I/O.

the problem when we don’t have IEN flag is that the I/O module reads data from the teletype but when the teletype has previous encoded data which is staying in it. But the CPU asks for input the module reads from it which has garbage value in it. Employing an IEN flag solves this problem. When an interrupt is generated the CPU stops the current tasks and works on the new task.

4. Yes, it is possible that we want to allow a process to wait on more than one event at the same time. It is prominent in Multi-thread process and inter process communication.

We know a given process can have more than 1 thread. So if we assign each thread different tasks like I/O operation and creation of child process then it is possible that a given process can be in more than one state.

8. Comparison of reading a file using single-threaded file server and a multi-threaded server.

Given that it takes 12 msec to get request for work, dispatch it and do the rest of necessary processing. So, in total it is doing 3 works for a given request.

1) For a single threaded the number of requests/secs it can handle is

When the data is in the cache (2/3 time)

1/ (12 ms) = 0.0833 * 10^3 = 83.33 which is around 83 requests.

When the data is not in the cache (1/3 time)

Then to handle a request it takes additional 75 msec. so in total the time to handle a request is (12+75) = 87 msec.

So the number of requests/sec it can handle is

1/(87 ms) = (1/ 87) * 10^3 = 11.494

Which is around 11 requests.

So, in total for a mix of request

(1/3) * 11 + (2/3) *83 = 58.9999 which is around 59 requests/sec.

2) for multi-threaded the number of requests/secs it can handle is dependent on the number of threads it had in it.

(Assumption that there are sufficient number of tasks greater than the requests we get)

So, if we get n requests which is a mix

(1/3) * 11n + (2/3) * 83 n = 58.9999 which is around (59 * n) requests/sec.