Uppsala University

Department of Computer Systems (DoCS)

IT3 (Datorsystem II : (Del 2))

Final Examination

Dec 2001

14:00 - 19:00
PostScriptum Sal 2

INSTRUCTIONS TO CANDIDATES

- This is a FIVE (5) hour examination
- Answer questions in English, Swedish or French.
- Dictionaries are Permitted
- The exam consists of 50 points, 25 pts in each section.
- Grading U (0-29), G (30-39), VG (40-50)

Section A : Parallel and Distributed Systems

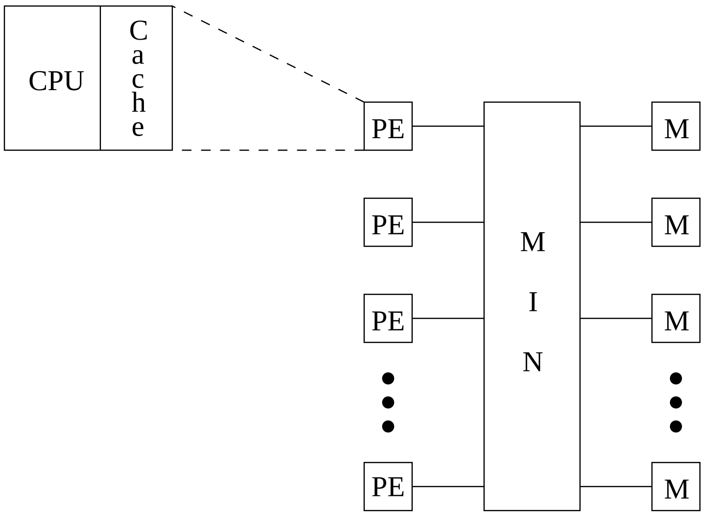
Question 1

Give an example of a computer architecture that falls into the MIMD class of computers. Draw a high level block diagram of the architecture and explain why it is an MIMD computer.

[4]

SOLUTION

An example of an MIMD computer architecture is a uniform memory access delay shared memory computer. Such computers are constructed from a number of powerful processing nodes (often called processing elements or PE's) which are connected to memory modules via a high speed uniform access delay network. These machines are most often constructed with a multistage interconnection network (MIN) as the core data transmission fabric, processing elements typically comprise a CPU, cache memory and perhaps a larger slower access speed local memory unit. A diagramatic representation of the general structure of such a machine is found below.



Question 2

Write at most one page discussing the two distributed programming langugages we have studied (C-Linda and CSP) commenting on the similarities and differences between them.

[4]

SOLUTION

This question is not relevant to the course in 2002.

Question 3

Distinguish between data partitioning and load balancing. What are they, and why is each of these things important when designing and implementing high performance programs?

[3]

SOLUTION

This issue was discussed in the lectures in 2002, but with slightly less weight than in 2001. The solution outline is below.

The chief difference between load balancing and data partitioning is apparent from the definitions of the two terms.

load balancing: *The objective of load balancing is to ensure that all processing elements in a parallel or distributed computer complete execution at the same instant. By allocating work in proportion to the characterstics of each element of the parallel or distributed machine we try to ensure that all processing elemnets complete execution simultaneously.*

data partitioning: *Here the objective is to divide up the data structures that are to be processed in such a way that access to the data by the concurrently executing processing elements does not result in race conditions. That is, PEs are not reading a writing to the same memory locations in a given phase of the program execution. If multiple PEs are accessing a single memory location and some a reading and some are writing to that location the values actually read by each PE cannot be guaranteed to be the same in each execution of the program. This has implications in terms of the results produced by the program, and thus the correctness of the results.*

Clearly data partitioning and load balancing are related, in that we need to divide up work in order to balance load as well as make sure that conflicts over read/write access to memory do not lead to `non-determinism` *and thus unpredictable program results in consecutive executions of the program. To make sure that program safety is ensured we can eliminate conflict over data structures using other techniques such as conditional critical regions and semaphores, however allocation of work to PEs remains a performance issue.*

Question 4

What is "non-determinism"? Determine using an interleaving diagram if non-determinism is present in the following program.

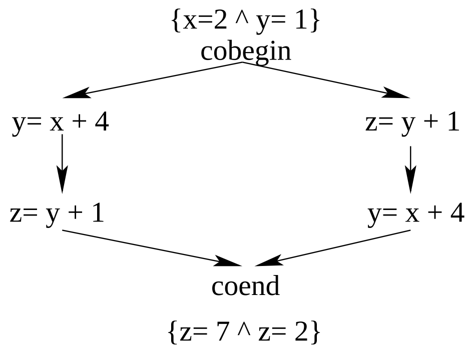
```
x= 2;
y= 1;
cobegin
    y= x + 4;
■
    z= y + x;
coend
```

[5]

SOLUTION

Non-determinism refers to the inability to predict the outcome of the execution of concurrent code segments due to read/write conflicts (otherwise known as `race conditions`*). This means that the same program can conceivably produce different results in consecutive executions given identical input data.*

For the given code fragment the interleaving diagram is below.



The interleaving diagram clearly demonstrates that the stored value of `y` *depends on whether the assignment of* `x+4` *to* `y` *occurs before the read of* `y` *in the other concurrently executing statement* `z= y + x`*. This means that the value of* `z` *can be assumed to be either 3 or 7 after execution of the code fragment given in the question.*

Question 5

Assume that you can create 1..k parallel copies of a C function using the command $ProcName(1..k, < params >)$; Each procedure copy can assume the following formal definition for its interface.

void ProcName(int Id, int x, int A[ ], int N, int k);

Where, Id is bound to a unique value in the range 1..N and serves as the identity of the copy of the procedure.

Using this parallel programming language structure write a parallel function to search through all the values in an array A to find the location of a desired value x. As soon as a value is found that is equal to x the procedures can stop executing. This means that if a matching value is found immediately all the processes should terminate, and most of the vector will remain unprocessed.

The formal definition of the function that you should write is as follows.

void FindNum(int Id, int x, int A[ ], int N, int k);

Where A is the Array of values to be searched, N is the number of entries in the array, k is and the number of processes. Assume that N is divisible by k, and that x is the number that is to be located, if it is found in the array A the index at which it is found should be returned.

[9]

SOLUTION

Assume the existance of some extra global variables, and that the programming model to be used is the "shared memory" model. This allows all the function copies to see two important global variable namely `int done, location;`

done *is used to allow all the copies to terminate as soon as any of them locates the value* `x`.

location *records the position (index value) where* `x` *occurs in the array* `A`.

Pseudo-code:

```
determine my area of A
while not done
    if A[index] is equal to x
        // OK to record x as found and exit
        location <- index
        done <- true
        return from function
    endif
    advance index to next value
endwhile
```

This can be implemented as the following code. In order for the following solution to be efficient we must assume a homogeneous parallel architecture, since the data partitioning scheme allocates equal work effort to each copy of the function. I also assume that the call to the function allocates Id's in the range [1..k] as indicated in the question.

NOTE: This code operates safely despite the fact that the updates to `done` *and* `location` *occur without locking. This is OK since multiple updates will always result in done=true and location being set to a location where x is found, and this is a safe termination of the program in all situations.*

```
void FindNum(int ID, int x, int A[ ], int N, int k)
```

```
{
    int start= (Id-1)*N/k;
    int stop= start + N/k;
    for(int index=0; index<stop && !done; index++) {
        if(A[index] == x) {
            location= index;
            done= 1;
            return;
    }
}
```

Section B : Realtime Systems

Question 6

Write a paragraph explaining what the term "deadline monotonic scheduling" means?

[2]

SOLUTION

This term describes a system in which the highest priority task is always executing and the priorities for tasks are allocated such that priority is allocated on the basis of the deadline. That is the shorter the deadline for the task the higher its priority will be in the system.

Question 7

Describe the operating data structures and activities associated with implementing an event driven scheduler. Using these structures and activities reason about the delays experienced by a task due to the activity of the scheduler and extend the following equation to include the costs associated with the scheduler execution costs.

$$R_i = J_i + \omega_i$$

$$\omega_i = C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{\omega_i + J_j}{T_j} \right\rceil C_j$$

[8]

SOLUTION

The solution involves understanding the additional overheads associated with timer expiry and moving processes from a buffer queue to the run queue. The approach is to postulate a set of high priority OS tasks with higher priority than all user tasks which carry out the activity associated with timer expiry.

In this case all the OS processes will have the same periods as the corresponding task for who's timer they are responsible, and the cost of executing in the OS can be represented by a factor C_event for instance. If all the OS tasks can preempt the user tasks since they have highest priority the factor we are looking for is related to the current preemption term in the equation given above. A simple manipulation gives the following additional term to be added into the analysis

$$\sum_{\forall k \in all\_tasks} \left\lceil \frac{\omega_i}{T_k} \right\rceil C_event$$

Question 8

Define each of the following in a single paragraph. You may use diagrams and pictures if you wish to.

- Rate monotonic
- Priority inversion
- Polling jitter
- Blocking time

[4]

SOLUTION

General definitions of these ideas are in the notes, and should be easily looked up in the online text, so there is no real need to repeat them here. You should know about these ideas, and probably illustrate your solution to the last three of these with a diagram to help explain the situation.

Question 9

Explain the operation of the CAN bus. In particular explain the CAN bus media arbitration algorithm.

Why is the CAN bus suitable for realtime applications in contrast to other common communication protocols such as Ethernet and Token Ring?

[4]

SOLUTION

The CAN bus uses a binary AND-gate access arbitration algorithm, in which all the messages that are ready to be sent at the point where the bus becomes free begin immediately to send their priority bits in perfect synchrony (lock-step). After sending each priority bit all bus interfaces test the state of the bus, which will be one if all interfaces sent a one, or zero if any process sent a zero. If you wrote one and read back zero you know that you cannot have the highest priority, so you back off and stop sending. The message that gets to send all its priority bits knows that it has the highest priority and proceeds immediately to send the payload of its message.

The CAN bus is suitable for realtime applications because it permits us to analyse the maximum queuing delay that any message can experience, as well as the upper bound on the time that a message can take to be delivered (based on the maximum payload size and bit stuffing situation for a worst case maximum size payload). This allows us to establish a firm upper bound on the delivery time required for each message in the system in the worst case. This response time can then be used in conjunction with holistic scheduling methods to analyse distributed realtime systems.

Question 10

Write three one paragraph explanations for important issues that arise in computing Worst Case Execution Times (WCET) for tasks in a realtime system.

[3]

SOLUTION

When computing WCET for if statements we compute the execution times for both branches and add the execution time for the longest branch to the execution cost of testing and branching to labels.

If we are to compute a WCET for a loop we need to know the worst case number of iterations that the loop can execute. This is a difficult situation, since in the worst case this might be the size of the machine representation of some index variable. An even worse situation arises when the termination condition for the loop is based on a boolean comparison with user input. In this case there is no way to analyse the code to determine the maximum number of loop iterations, and programmers need to supply analysis information in terms of invariants that are specified in the code to allow automated analysis.

Worst case execution times for some instructions such as multiplication depend on the size of registers and the number of set bits in the operands. In this case we also need to select the worst case bit patterns that cause the longest micro-code executions in the CPU.

Compute response times for the following system of tasks and determine if the system is schedulable given that deadline monotonic priority assignment is used.

P	Task	T	D	C
1	A	35	35	28
2	B	56	50	15
3	C	128	123	34

HINT:

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

[4]

SOLUTION

Standard analysis indicates pretty quickly that this system has the following response times R_A *=28 ,*R_B *=99 , and* R_C *= Infinite. Thus the system is not schedulable as it is currently designed.*