[Template for Applied Session submission]

Applied Session Week {Week 4}

Student ID	Student Name	Student Email
32837933	Chua Sheng Xin	schu0077@student.monash .edu

Tasks

{Answers

Question 1
A. The types of parallel systems based on the number of independent instruction and data streams are SISD, SIMD, MISD and MIMD. SISD or Single Instruction Stream Single Data is a computing system that operates on sequential execution of a single instruction stream on a single data stream and it operates with only one core. SIMD or Single Instruction Stream Multiple Data Stream is a computing system that operates by performing a single instruction stream on multiple data streams. This system is created out of the necessity that the same operation needs to be perform repeatedly on multiple data streams. MISD or Multiple Instruction Stream Single Data Stream is a computing system that executes operations of multiple instruction stream on a single data stream. MIMD or Multiple Instruction Stream Multiple Data Stream is a computing system that operates on parallel execution of multiple instruction streams on multiple data streams. The system is the most modern and the most common of all computing systems.
B. An example of the SISD system is the Von Neumann Machine, however it can also be found in other simple hardwares for example, mircrowave and oven. These hardwares often only require simple execution of instruction on a single data stream. An example of the SIMD system is the Vector Machine, for example Cray 9000. The SIMD is able to perform the same operation over and over for multiple data streams hence it is useful for vector operations that require repeating the same operations with different input of data. An example of the MISD system is the decryption machines that tries different decryption algorithms on a single encryption, outside of that it is not widely used. An example of the MIMD system is Intel and other major information technology companies as it is used by most modern computers and is by far the most common computing system.

Question 2
A. The composition of the process involves various attributes that allows it to be identified and then executed. These attributes are identifier, state, priority information, program counter, memory pointer, context data, I/O status information and accounting information.
B. The two resources that are shared between threads within a process, but not shared between processes are memory space and file descriptors. Threads within the same process share the same memory space, including the heap and global variables. This allows threads to access and modify the same data, facilitating communication and synchronization. However, memory space is not shared between different processes, which each have their own separate memory space. Additionally, threads within a process share the same set of file descriptors, which means they can access the same files, sockets, and other I/O resources. This is unlike processes, which have their own independent set of file descriptors.

C. The three IPC mechanisms applicable to the same host operating system are shared memory, message passing and Unix signals and analogues. Shared memory involves 2 or more processes sharing the same memory space and reading and writing into the memory pages. Message passing requires processes to send and receive messages to each other, has different protocol between operating systems and is usually quite slow and cumbersome. Lastly, Unix signals and analogues are used by user or super user to control process state. An example is the Kill Minus Nine command on Linux to kill a process.
D. The IPC mechanism that is applicable for a network of computers is BSD Socket mechanism.
E. The two aspects of performance for BSD Socket mechanism are throughput and latency. Throughput refers to the volume of data that is sent over link and is measured by Gigabits/s or Gigabytes/s. Latency refers to the time delayed that occurs when the data is being passed from the source process to the destination process. These two aspects are often related and improving one can lead to improvements in the other. However, due to the design of network and IPC scheme in use, even high bandwidth networks can have high latency. Moreover, where data dependencies exist, latency is a problem.

Question 3
A. The possible outcomes of the code above are if both of the threads print successfully and one or both of the threads did not print successfully. In this scenario, both threads execute successfully, and the messages "Thread 1" and "Thread 2" are printed as expected. The order in which the messages are printed is not guaranteed because the thread scheduler may allow either thread to run first. Both iret1 and iret2 return 0, indicating that the threads were created successfully.
Thread 1 returns: 0
Thread 2 returns: 0
Thread 1
Thread 2
In the scenario where one or both threads fail to print, the below may show. This happens when the main thread finishes executing and exits before the child threads complete, the program may terminate, preventing one or both threads from printing their messages. This happens because the program does not wait for the threads to complete before exiting. However the main thread still shows 0 because it successfully created the thread even though it did not finish printing the message.
Thread 1 returns: 0
Thread 2 returns: 0
B. The reasoning behind the outcomes is that the code creates and runs two independent threads. The threads execute the same function, print_message_function, but with different arguments, message1 and message2. When the code runs successfully and both threads prints out the message assigned for them then it will show 0 for the successful creation of threads and message1 and message2 after. This is because the messages can only be printed after the threads are created. The order of the threads may vary due to how the threads are scheduled. If the code does not run successfully and the threads do not print the messages, the messages are not displayed. However, the line showing 0 will still show if the threads are created successfully.

Question 4
A. The diagram without pipelining:

Timephase (Nanoseconds)	Stage 1	Stage 2	Stage 3	Stage 4
1	Instruction N			

2		Instruction N		
3			Instruction N	
4				Instruction N

The diagram with pipelining:

Timephase (Nanoseconds)	Stage 1	Stage 2	Stage 3	Stage 4
1	Instruction N			
2	Instruction N+1	Instruction N		
3	Instruction N+2	Instruction N+1	Instruction N	
4	Instruction N+3	Instruction N+2	Instruction N+1	Instruction N
5		Instruction N+3	Instruction N+2	Instruction N+1
6			Instruction N+3	Instruction N+2
7				Instruction N+3

B. The throughput speedup and the pipeline latency for such a pipeline system is four times the system without pipelining and 4 nanoseconds. This is because the pipeline system utilizes the processing elements to run without any idle stages so that each instruction runs concurrently. Hence the pipeline system is able to compute one result per time phase compared to one result per four time phase of the non-pipelining system. As for pipeline latency, the time it takes for a single instruction to pass through all stages of the pipeline is 4 nanoseconds, as each stage takes 1 nanosecond. It is the same as the pipeline without latency.
C. The two reasons for pipeline stalls are data dependency and conditional branches. Data dependency refers to the system having to wait for the result of operation N while operation N+1 is ready to execute because operation N+1 requires the result of operation N. Moreover, the system has to delay at least 1 timephase until it can execute instuctions again. This causes stalling in the pipeline due to the fundamental structure of the algorithms. Conditional branches are another reason for pipeline stalling as the pipeline system cannot determine which branch to execute until the result for the branches is returned. The whole pipeline is stalled as the system has to finish computing the results to decide which branch to continue.

Question 5
A. Time phase diagram:

Timephase (Nanoseconds)	Stage 1	Stage 2	Stage 3	Stage 4
1	Instruction 1,2			
2	Instruction 3,4	Instruction 1,2		
3		Instruction 3,4	Instruction 1,2	
4			Instruction 3,4	Instruction 1,2
5				Instruction 3,4

Given 4 integer instructions, there are 2 integer Executions Units to handle the instructions. Each Execution Unit is able to handle one instruction and 2 Executions Units are able to execute 2 integer instructions simultaneously, with pipelining the Execution Units are able to continue executing the last two instructions in the next time phase. The total time phase needed is 5 nanoseconds because the two Executions Units run simultaneously and concurrently in each time phase to return the result of the 4 integer instructions.
B. Anti-dependency occurs when an instruction depends on a previous instruction writing to the same register or memory location, but the order of execution does not align with the order of instructions due to the pipeline's reordering or parallel execution. Anti-dependency

can cause pipeline stalls or delays if instructions that write to the same register or memory location are executed out of order. This is because the CPU must ensure that the value written by an earlier instruction is not overwritten or invalidated by a later instruction.
Example in C/C++:

```
#include <iostream>
int main() {
    int a = 10;
    int b = 20;
    // Instruction 1: a = a + b
    // Instruction 2: b = b + 1
    a = a + b; // (Write to 'a')
    b = b + 1; // (Write to 'b')
    // Instruction 3: a = a + b
    // Instruction 4: b = b + 1
    a = a + b; // (Write to 'a')
    b = b + 1; // (Write to 'b')
    std::cout << "a = " << a << ", b = " << b << std::endl;
    return 0;
}
```

If instructions 1 and 2 are executed out of order or in parallel, anti-dependency might occur. For instance, if instruction 3 executes before instruction 2 finishes updating b, the result of a = a + b in instruction 3 might use an outdated value of b, leading to incorrect results.}

{Screenshots}

{References
I have used ChatGPT to generate sentences and codes for this assignment, especially in Question 5.}