True about shared memory & message passing?- shared memory may be slower than message passing on multi-processor system.

Not true about message passing?- in indirect communication, multiple links may exist between pair of process.

Spinlock – are good for multiprocess b/c the waiting can be on one &rest can be on other.

Mutex lock- waiting process is replaced in queue. State: wait lock() checks on value of lock, if available enters oth erunise waits unlock().

Spinlock- waiting process is busy waiting (still gets cpu cycles). State: running/ready this makes more sense while wait time its smaller.

Dual model - when process does something wrong the hardware will take control over the operating system.

Monitor - are by themselves not enough powerful to solve synchronization problems condition x,y.

Data parallelism- distribute subset of same data across multiple cores, same operation on each.

Task parallelism- distributing thread across cores each thread performing unique operations.

Starvation- process in not guaranteed to be remove from semaphore wait list in which it suspended. Solution: bounded waiting. I turn per process not turns. It eventually get turn in starvation where as in deadlock no change.

Priority inversion- lower priority process holds a lock needed by higher priority process. Solved by priority: inheritance protocol.

Diners problem: (i+1)%5 or (i-1)%5 1. Allows only 4 philosophers to be sitting simultaneously on the table. 2. Allow a philosopher to pick up only if both are available (use critical section) 3. Use an asymmetric sol, odd: pick left before right. Even: pick right before left.

Idealization assumptions for AMDAHL’S LAW: 1. Ignore OS overhead creation, context switching maintenance.

2. ignore communication & synchronization. 3. Perfect balance is achieved in division of programs into n.

3 reasons that causes a process to transition from running to wait: 1. Process need to perform I/O operations. 2. Waiting for some event to occur such as a non busy, waiting semaphore. 3. Time quantum expires.

Different reasons that causes process to transition from running to waiting: 1. Requesting I/O. 2. Waiting for a children to terminate. 3. Waiting for a semaphore.

1. What is (are) the advantage(s) of dividing an application into multiple threads relative to dividing it into multiple processes? –BOTH: Using less resources. - Easier communication using global variables
2. In a given program, only one sixth of the code is parallelizable. What’s the maximum speedup factor that can be achieved on a dual-core system under ideal conditions? – 1/(5/6+1/6) = 12/11
3. How does protecting a critical section (CS) with a semaphore ensure mutual exclusion? - When a process is in its CS, any process that tries to access a CS that is protected by the same semaphore is placed in the waiting state.
4. Which of the following is true about spin locks and mutex locks? –BOTH: A process waiting on a spin lock uses CPU cycles but a process waiting on a mutex does not. - Using mutex locks involves more context switching.
5. Which of the following is **not** true about shared memory and message passing? - Shared memory is slower than message passing on all multiprocessor systems.
6. Which of the following is **not** true about Remote Procedure Calls (RPCs)? - The user program communicates directly with the matchmaker without kernel intervention.
7. What is the maximum number of processes/threads that can be active in a **monitor** at the same time? – One.
8. What is the difference between and I/O-bound process and a CPU-bound process? -
9. What are the limitations of Amdahl’s Law? – BOTH: It assumes that the parallelizable code can be divided equally among the CPUs. – It does not account for communication and synchronization overhead.
10. Which of the following is true about threads? –
11. Which of the following is true about process states? –
12. Which of the following is **not** true about an **Orphan** process? – Its parent has not called wait(), but the parent has not terminated yet.
13. How does protecting a critical section (CS) with a semaphore ensure mutual exclusion? - When a process is in its CS, any process that tries to access a CS that is protected by the same semaphore is placed in the waiting state.
14. Which of the following is **not** true about shared memory and message passing? - Shared memory is slower than message passing on all multiprocessor systems.
15. Which of the following is **not** true about message naming? - In direct communication, multiple links may exist between a pair of processes.
16. Which of the following is **not** true about Remote Procedure Calls (RPCs)?- The user program communicates directly with the matchmaker without kernel intervention.
17. Which of the following is **not** true about concurrency and parallelization? - With concurrency, it is impossible achieve any speedup relative to sequential execution.
18. In a given program, only one fifth of the code is parallelizable. What’s the maximum speedup factor that can be achieved on a quad-core system under ideal conditions? - 20/17.
19. which of the following is true about shared memory and message passing? - Shared memory may be slower than message passing on multiprocessor systems.
20. void Producer(int bufSize, int itemCnt, int randSeed){
21. int i, in = 0, out = 0, val;
22. 1 for (i=0; i<itemCnt; i++) {
23. 2 while((GetIn()+1)%bufSize == GetOut());
24. 3 val = GetRand(0, 1000);
25. 4 WriteAtBufIndex(in, val);
26. 5  **X1** in = (in + 1)%bufSize;
27. 6 SetIn(in);
28. }
29. }
30. void Consumer(){
31. //Code to open shared memory block and map it to gShmPtr
32. 1 int bufSize = GetBufSize();
33. 2 int itemCnt = GetItemCnt();
34. 3 int in = GetIn();
35. 4 int out = GetOut();
36. 5 for(i=0; i<itemCnt; i++){
37. 6 **X2** while(GetIn() == GetOut());
38. 7 val = ReadAtBufIndex(out);
39. 8 **X3** out = (out + 1)%bufSize;
40. 9 SetOut(out);
41. }

* Which line in the above code does each of the following? Make sure you specify whether that line is in the Producer or in the Consumer

Waiting if the buffer is empty- Consumer wait when buffer is empty.

Updating the value of *in* in the shared memory buffer- line 6 in producer update.

* The length of the time quantum (slice) given by the operating system scheduler to each process must be selected carefully. What’s the negative consequence of making this time slice too long and what’s the negative consequence of making it too short?

Problem with too long time quantum- the system is less responses.

Problem with too short time quantum- a lot of time will be waist in context switching.

1. Give two clear advantages of using multiple threads compared to using multiple processes.

1. Faster, because there is less system overhead during creation and context switching

2. Resource sharing. Since threads share resources, that better utilizes system resources.

3. Faster communication using global variables

1. Name three **different** reasons that may cause a process to transition from the running state into the waiting state.

1. Requesting I/O

2. Waiting for a child to terminate

3. Waiting for a semaphore

1. Amdahl’s Law gives the potential performance gain that may be achieved by using multiple processors under **ideal conditions**. What are the ideal conditions that must be satisfied in order to actually achieve the performance gain computed by Amdahl’s Law?

1. There is zero overhead. So, we are ignoring thread (process) creation and context switching overhead as well as communication and synchronization overhead.

2. We are assuming perfect load balancing (we can perfectly divide the work among multiple threads or processes).

|  |  |
| --- | --- |
| Threads | Process |
| * Run in a shared memory space. * Allocating space for threads are user space. * Scalability process can take in advantages if multiprocessor architecture. | * Process run in separate memory space. * Kernel space * Hierarchy weight * Less resources sharing * Economy & speed |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Change | Will it work correctly? | Items  Produced | Items  Consumed | Explanation |
| Delete ***SetIn()*** on Line 6 of the Producer | No |  |  |  |
| Replace ***GetIn()*** with ***in***  ***Producer***  ***consumer*** | Yes  No |  |  |  |
| Replace **GetOut()** with ***out***  Producer  consumer | No  Yes | n | n | Out is update by consumer. Doesn’t need share memory. |

|  |  |  |  |
| --- | --- | --- | --- |
| Case | Max useful producers | Max speedup ratio | Brief Explanation |
| ConsTime = ProdTime | 1 | 1 | Time = speeding up prod will not help |
| ConsTime = 5 x ProdTime | 1 | 1 |  |
| ProdTime = 5 x ConsTime | 5 | 5 | Speed up to 5 times of producer |