**Tutorial 5: Concurrent Processes**

Q1. Write short note on “parallel processing”.

|  |
| --- |
| * parallel processing refers to 2 or more processor are operating simultaneously at RUNNING state to produce a result * Processor Manager has to coordinate activity of each processor and sync interaction among CPUs. |

Q2. In a system consisting of some concurrent processes running asynchronously, the problem of critical section/region must be addressed.

1. Define the critical section problem.

|  |
| --- |
| * The critical section problem refers to the problem of how to ensure that at most one process is executing its critical section at a given time. |

1. Figure 1 depicts an excerpt of a code. Identify the critical section and the remainder section using the line numbers given.

|  |  |
| --- | --- |
| 1  2  3  4  5 | A=1  B=2  C=A+B  print(“The sum of A and B is”, C)  print(“All the best for your exam!”) |

|  |
| --- |
| Critical section: Line 3  Remainder section: Line 4,5 |

1. Some operating systems use semaphores to provide for process synchronization. Define the ***wait*** and ***signa***l operations for a binary semaphore. You should provide the pseudocode to illustrate the purpose of each operation.

|  |
| --- |
| **wait ()**  So this operation is to check if there are any other process accessing the critical section currently . If yes, S.value would be negative, and the process issuing wait() would be blocked, to ensure that the mutual exclusive access to the critical section.  wait (S){  S--;  if (value *<* 0) {  *add this process to waiting queue*  block();  }  }  **Signal()**  Upon exiting the critical section, a process will be issuing signal(), check if there were other processes waiting on the queue. If yes, a process is then removed from the blocked state, and allowed to continue its execution.  Signal (S){  S++;  if (S<= 0) {  remove a process P from waiting queue  wakeup(P);  }  } |

1. Will semaphore operations be in deadlock situation? Justify your answer.

|  |
| --- |
| Incorrect use of semaphore operations will still cause deadlock occur, if perform signal system is called before wait system call or two wait system of two different semaphore value or omitting of wait or signal or both. |

Q3. The following program consists of 3 concurrent processes and 3 binary semaphores. The semaphores are initialized as S0=1, S1=0, S2=0.

|  |  |  |
| --- | --- | --- |
| Process P0 | Process P1 | Process P2 |
| while (true) {  wait(S0);  print (0);  signal(S1);  signal(S2);  } | wait(S1);  signal(S0); | wait(S2);  signal(S0); |

How many times will process P0 print “'0”? Justify your answer.

|  |
| --- |
| **At least twice**  **P0** will execute first because only **S0=1**. Hence it will print 0 (for the first time). Also P0 releases S1 and S2. Since S1=1 and S2=1, therefore P1 or P2, any one of them can be executed.  Let us assume that **P1** executes and releases S0 (Now value of S0 = 1). Note that P1 process is completed.  Now S0=1 and S2=1, hence either P0 can execute or P2 can execute. Let us check both the conditions:-  1. **Let us assume that P2 executes**, and releases S0 and completes its execution. Now P0 executes; S0=1 and prints 0 (i.e. second 0). And then releases S1 and S2. But note that P1 and P2 processes has already finished their execution. Now P0 starts its execution and again prints 0 (third 0) and releases S1 and S2 (Note that now S0=0). Again if P0 tries to execute it goes into sleep condition because S0=0. Therefore, minimum number of times '0' gets printed is 3.  *or*  2. Now, **let us assume that P0 executes**. Hence S0=0, (due to wait(S0)), and it will print 0 (second 0) and releases S1 and S2. Now only P2 can execute, because P1 has already completed its execution and P0 cannot execute because S0 = 0. Now P2 executes and releases S0 (i.e. S0=1) and finishes its execution. Now P0 starts its execution and again prints 0 (third 0) and releases S1 and S2 (Note that now S0=0). P1 and P2 has already completed its execution therefore again P0 takes its turn, but since S0=0, it goes into sleep condition. And the processes P1 and P2 which could wakeup P0 has already finished their execution. Therefore, maximum number of times '0' gets printed is 3. |

Q4. (a) Mutex and semaphore are kernel resources that provide synchronization services. Differentiate between semaphore and mutex.

|  |
| --- |
| * Mutex is a locking mechanism used to synchronize access to resource. Ownership associated with mutex. Only the owner can release the lock. * Semaphore is signaling mechanism (“I am done, you can carry on” kind of signal) .   Semaphore supports wait and signal operations modification, whereas Mutex is only modified by the process that may request or release a resource. Semaphore value is modified using wait () and signal () operations, on the other hand, Mutex operations are locked or unlocked. |

(b) Explain the bounded buffer producer-consumer problem and explain the use of semaphore in solving this problem.

|  |
| --- |
| * The **producer-consumer problem** occurs when two threads exist, one is “producing” data to store in the buffer and the other is “consuming” that data from the said buffer. * **Concurrency problems** arise as there is a need to keep track of the number of items in the buffer, which has a fixed limit on how many items can be inside it at any one time. * The producer-consumer problem can be solved by using **3 semaphores for each following purpose**:   + to Indicate the number of filled positions in buffer.   + To indicate the number of empty positions in buffer.   + Mutex, to ensure mutual exclusion between processes |

Q5. (a) The problem of readers and writers arises when 2 types of processes need to access a shared resource. Discuss two solutions using P and V operation.

|  |
| --- |
| Readers can inspect items in the buffer, but cannot change their value. Writers can both read the values and change them. The problem allows any number of concurrent reader threads, but the writer thread must have exclusive access to the buffer.   1. **Give priority to readers** over writers so readers are kept waiting only if a writer is actually modifying the data. However, this policy results in writer starvation if there is a continuous stream of readers. 2. **Give priority to the writers**. In this case, as soon as a writer arrives, any readers that are already active are allowed to finish processing, but all additional readers are put on hold until the writer is done.Obviously this policy results in reader starvation if a continuous stream of writers is present |

## (b) Provide a situation in which the Readers-Writers Problem can occur in a computer system.

|  |
| --- |
| A typical example of this occurs with databases, when several processes are accessing data; some will want only to read the data, others to change it. The database must implement some mechanism that solves the readers-writers problem. |

Q6. Rewrite the following arithmetic expressions to take advantage of concurrent processing by using the terms COBEGIN and COEND to delimit the sections of concurrent code.

|  |  |
| --- | --- |
| 1. G= (A+C2) \* (E-1)3 / (D + B) | 1. H=A\*B\*C+990/(D+E)\*(F-G) |
| *Cobegin*  *T1 = C \* C*  *T2 = E - 1*  *T3 = D + B*  *Coend*  *Cobegin*  *T4 = A+T1*  *T5 = T2 \* T2*  *Coend*  *T6 = T5 \* T2*  *T7 = T4 \* T6*  G = T7 / T3 | *Cobegin*  *T1 = A \* B*  *T2 = D + E*  *T3 = F - G*  *Coend*  *Cobegin*  *T4 = T1 \* C*  *T5 = T2 \* T3*  *Coend*  *T6 = T4 + 990*  *H = T6 / T5* |

Self-Review

Q1. In the context of concurrent programming, differentiate between explicit parallelism and implicit parallelism.

|  |
| --- |
| Explicit parallelism: a type of concurrent programming that requires programmer to explicitly state which instructions can be executed in parallel.  Implicit parallelism: a type of concurrent programming that the compiler can detects which instructions can be performed in parallel automatically. |