Quizlet

IT OS Concept Ch5

Study online at quizlet.com/_1zpsyw

1.	**Addressing**	Schemes for specifying processes in send and receive primitives fall into 2 categories. Direct addressing and indirect addressing.
2.	**Blocking Send, Blocking Receive**	Both sender and receiver are blocked until message is delivered. Aka rendezvous. Allows for tight synchronization between processes.
3.	**Compare&Swap Instruction**	Aka compare and exchange instruction. A compare is made between a memory value and a test value. If the values are the same a swap occurs and its carried out atomically.
4.	**Concurrency arises in 3 diff contexts**	Multiple applications, structured applications, OS structure.
5.	**Consequences**	Don't know if a process will be blocked or not, don't know if process will continue immediately on a uniprocesor system when 2 processes are running concurrently. don't know if another process is waiting or not
6.	**Difficulties of Concurrency**	Sharing of global resources, difficult for the OS to manage the allocation of resource optimally, difficult to locate the programming errors as results are not deterministic and reproducible.
7.	**Direct Addressing**	Send primitive includes a specific identifier of the destination process. Receive primitive can be handled in one of two ways: require that the process explicitly designate a sending process, and implicit addressing.
8.	**Disadvantages**	Busy waiting is employed using processor time, starvation is possible, dead lock is possible.
9.	**Implementation of Semaphores**	semWait and semSignal operations be implemented in hardware or firmware. Dekker's or Peterson's algorithms can be used. Use a hardware supported schemes for mutual exclusion.
10.	**Indirect Addressing**	Message sent to shared data structure consisting of queues that can temp hold messages, queue aka mailboxes, one process sends message and other picks up from mailbox, allows for greater flexibility.
11.	**Message Passing**	Send & Receive. A process sends info in a message to another process designated by a destination. Process gets info by executing the receive primitive, indicating the source and the message.
12.	**Message Passing**	Process interaction requires 2 things: synchronization and communication. Message passing can provide both of these functions.
13.	**Monitor Characteristics**	Local data variables are accessible only by the monitors procedures and not by any external procedure. Process enters monitor by invoking one of its procedures. Only one process may be executing in the monitor at a time.
14.	**Monitors**	Programming lang. constructs that provide equal functionality to that of semaphores and is easier to control. Used in many languages. software module consist of one or more procedures, an initialization sequence, and local data.
15.	**Multiple Processes**	OS design is concerned with the management of processes and treads: multiprogramming, multiprocessing, distributed processing.
16.	**Mutual Exclusion: Hardware support**	Interrupt Disabling: uni-processor system, disabling interrupts guarantees mutual exclusion. Disadvantages: efficiency of execution could be noticeably degraded, this approach will not work in w multiprocessor architecture.
17.	**Non blocking send, blocking receive**	sender keeps going but receiver is blocked until request message arrives. Most useful combination. Sends many messages to a variety of destinations quick as possible.
18.	**Non blocking send, nonblocking receive**	Neither party is required to wait
19.	**OS concerns**	Be able to keep track of various processes, allocate and de-allocate resources for each active process, protect the data and physical resources of each process against interference by other processes, ensure that the processes and output are independent of the processing speed.
20.	**Principles of Concurrency**	Interleaving and overlapping. Uniprocessor - relative speed of execution of processes cannot be predicted.

21.	**Producer/Consumer Problem**	The problem: ensure that the producer can't add data into full buffer and consumer can't remove data from an empty buffer.
22.	**Race Condition**	Occurs when multiple processors or threads read and write data items. Final result depends on the order of execution.
23.	**Readers/Writers Problem**	A data area is shared among many processes, some only read and some only write. Conditions that must be satisfied: any number of readers can read at the same time, only one writer at a time, and when writing it cant be read.
24.	**Requirements for mutual exlusion**	Must be enforced, no dead lock or starvation, process with halts must do so without interfering with other processes, process must not be denied access to critical section when there is no other process using it, no assumptions are made about relative process speeds or number of processes, a process remains inside its critical section for a finite time only.
25.	**Resource Competition**	Processes come into conflict when they are competing for resources. In some cases 3 control problems must be faced: need for mutual exclusion, deadlock, and starvation.
26.	**Semaphore**	Variable that has an integer value upon which only 3 operations are defined: there is no way to inspect or manipulate semaphores other than these 3 operations. May be initialized to a non negative integer value. The semWait operation decrements the value. The semSignal operation increments the value.
27.	**Special Machine Instruction Advantages**	Applicable to any number of processes on either a single processor or multiple processors sharing main memory. Simply and easy to verify. Can be used to support multiple critical sections; each critical section can be defined by its own variable.
28.	**Strong/Weak Semaphore**	Queue is used to hold processes waiting on the semaphore. Strong: process has been blocked the longest is released from the queue first. Weak: order in which processes are removed from the queue is not specified.
29.	**Synchronization**	Reached by the user of condition variables that are contained within the monitor and accessible only within the monitor. Operated on two functions: cWait and cSignal.
30.	**Synchronization**	Talking of a message between two processes. When a receive primitive is executed two things may happen: if no waiting message the process is blocked or the process continues to execute, leaving the attempt to receive.