WWW.KTUNOTES.IN

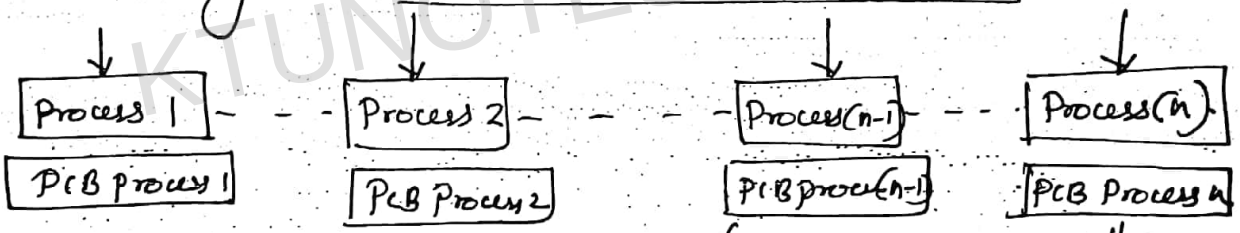
Module 5.

Inter Process Communication + Synchronization.

Process:

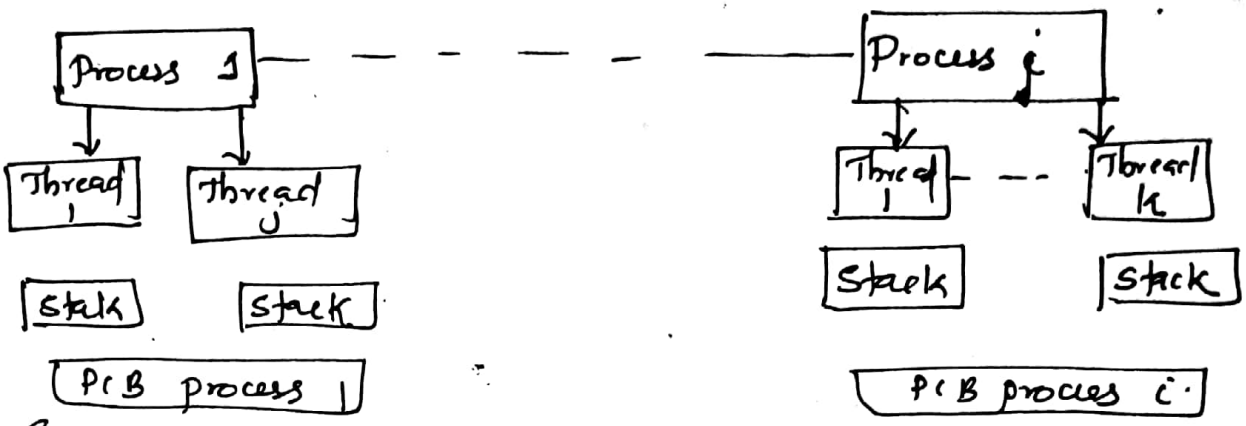
S/w consist of no. of processes and each process runs under the control of an OS. Processes are units of computation, execution of code in which are controlled by the OS, inter-process communication, resource-manager, s/m memory, s/m resources (n/w, file, display, printer) and access control mechanisms and are processed concurrently.

PCB → Process Control Block

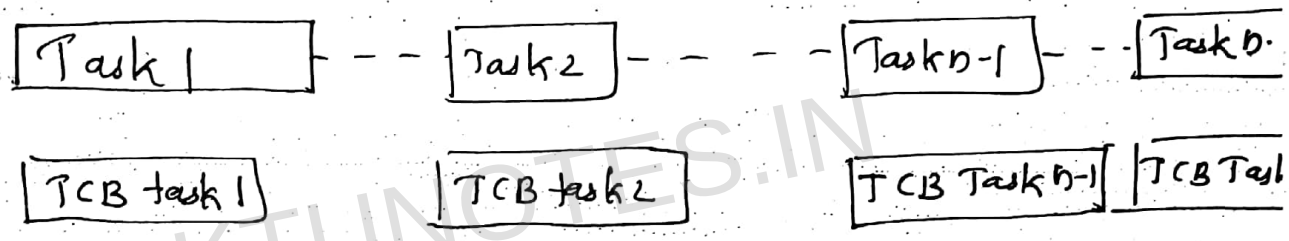


A thread is a process / sub-process within a process that has its own program counter, its own stack pointer, and stack, its own priority-parameter for its scheduling by a thread-scheduler and its own variables that load into the processor registers on context switching and is processed concurrently along with other threads.

KTUNOTES.IN
RIET
Downloaded from Ktunotes.in



Tasks are embedded program computational units that run on a CPU under the state-control using a task control block.



KTUNOTES.IN

(1) A process consists of an executable program (code) controlled by the OS.

(2) Process state means whether the process status is ready, running, blocked or finished and process structure means data, objects, resources and process control block (PCB)

(3) A process that runs when it is scheduled to run by the OS (kernel), which gives the control of CPU to a process including running

Downloaded from Ktunotes.in

a process on a requsf (system call)

Process is that unit of computation, which is controlled by OS scheduling mechanism, which lets it execute on the CPU. OS resource management mechanism lets the process use the s/m memory and other resources of the s/m, such as devices, I/Os, file, display et

Process is defined as a computational unit that processes on a CPU under the control of a scheduling kernel of an OS. It has a state defines by process-state and process-structure.

Process Control Block (PCB)

PCB is a data structure having the information of a process. The OS controls the process state using PCB. Only the process and OS can access PCB. PCB is stored in protected memory block. The OS protects the PCB from access by any other processes. Each process has a seperate PCB.

KTUNOTES.IN

Downloaded from Ktunotes.in

A PCB has following information about the process state.

(1) Process ID, priority, parent process (if any), child process (if any), address to PCB of next process which will run.

(2) Allocated pgm memory addr, allotted in physical memory and addresses in secondary memory for the instns of process.

(3) Allocated process-specific data memory addr.

(4) Allocated addr. for process heap, which means data generated during the process run.

(5) Allocated process-stack address for the funcn called during the process run.

(6) Allocated addr. for ## saving the CPU registers as a process-context which includes the program counter and stack pointer.

(7) Processes state Signal Mask. [when mask=0, (actn the process is inhibited from running and when reset to 1, the process is allowed to run].

(8) Alloted addresses of signals & IPCs dispatch tab

(9) OS alloted resources, descriptors [ex: file descriptor for open files]

(10) Security restrictions & permissions.

KTUNOTES.IN

Downloaded from Ktunotes.in

Process Context.

Present cpu registers, which include program counter and stack pointer are called context. Context saved on the PCB pointed process ~~task~~ stack and register save area addr. The running process stops after context saves and other process context now loads and that process runs. Context switching results in start of new process.

Multiple Thread in an Appln.

Embedded s/w can be said to consists of no. of threads or no. of processes and threads.

(1) A thread consists of executable program (codes) and has a state.

(2) The state of a task is represented by thread-status (running, blocked or finished), thread structure - its data, objects and a subset of the process resources, and thread-stack.

(3) A thread is a lightweight entity.

A thread may be a process level controlled entity and does not depend on the

KTUNOTES.IN

Downloaded from Ktunotes.in

kernel-level processes. A process is considered a heavyweight entity. It means process may depend on kernel level processes such as GUI, memory management unit. Process can have code in secondary memory from which the pages can be swapped into the physical primary memory during running of the process. The process therefore have process structure with entities such as virtual memory map, file descriptors and user-ID

A thread is defined as minimum computational unit that is allocated resources by the OS and that processes on CPU under the control of OS scheduler. Each thread has a state, which at an instance defines by thread-states (ready, running, blocked / finished), thread structure - its data, objects and resources and thread stack

KTUNOTES.IN

Downloaded from Ktunotes.in

Multiple Threads of a Process.

OS runs more than one process. When a process consists of multiple threads, it is called multithread process. A thread can then be consider as daughter process. A thread then define a minimum unit of a multithread process, to which OS Kernel allocates the s/w resources and provide a scheduling mechanism, which runs the thread on CPU. Different threads of a process may share common process-structure b/w the threads. Multiple threads can share the data of their parent process. Threads of a process can share the PCB information. It is light weight and does not depend on kernel level and other processes, such as GUIs and Memory management unit.

Multithread Programing.

An appln/process consists of multiple threads. A thread is a process or subprocess within a process that has its own program counter, stack pointer and stack, priority parameter

KTUNOTES.IN

Downloaded from Ktunotes.in

and has variables that load into the processes registers on context switching. It also has its own ~~kernel~~ signal mask at the OS kernel. Signal mask, when unmasked sets the thread active and run. When masked, the thread is put into a queue of pending threads. A thread stack is at a memory address block allocated by the OS. When a function in a thread in OS is called, the state information of calling function is placed on the stack top. When there is return, the calling function takes state information from the stack top.

KTUNOTES.IN

Pre-emptive and Non-pre-emptive

A thread scheduling mechanism can be pre-emptive and non-pre emptive. Pre-emptive means when a thread of higher priority becomes ready to run, then OS. preempts the running of low priority threads. and high ~~priority~~ threads executes on CPU. Non-pre emptive means when threads are considered equal priority, and when a thread finishes its allotted time slice then another thread executes on CPU.

Downloaded from Ktunotes.in

Tasks.

Task is the term used for the process in RTOS. A task is similar to process or thread in an OS. Some OS use the term task and some use the term process.

1) A task consists of sequentially executable pgm (code) and a state.
2) The state information in task is represented by task status (running, blocked or finished), task structure - its data, objects and resources and task control block (TCB).

An appln can also be defined as a program consisting of tasks and task behaviour in the various states. Task states are controlled by OS kernel process.

Task is defined as computational units that runs on CPU under the control of a scheduling kernel of an OS. OS controls the state, which at an instance is defined by task status, structure and control block.

KTUNOTES.IN

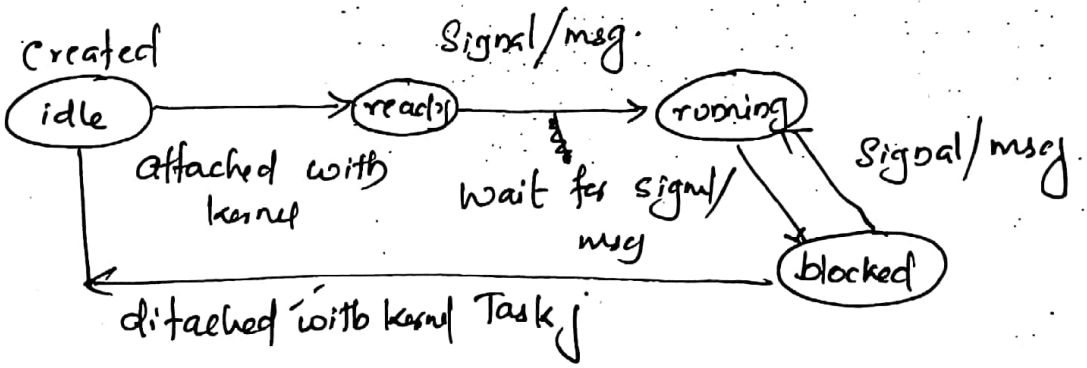
Downloaded from Ktunotes.in

Multitasking

An appln. may consists of no. of tasks and each task run needs a control of CPU given by the OS. Assume that there is only one CPU in a s/m. Each task is independent that takes control of CPU when scheduled by OS scheduler. The scheduler controls and runs the task. Task is an independent process. No task can call another task. The task can send signal(s) or message(s) that can let another task run. The OS can block a running task and allow another task to gain access of CPU to run the codes.

KTUNOTES.IN

Task and Thread States:



States of task/Thread

Task has a state, which includes its status at a given instance in the s/m. State can be one of the following: idle (created), ready,

Downloaded from Ktunotes.in

running, blocked and deleted (finished). Task is in ready state again after finish when it has infinite waiting loop. Multitasking operations are by context switching b/w various tasks.

Thread can either be a sub process or a process within an appln. A task is a process and the OS does the multitasking. Task is a kernel controlled entity where thread is a process controlled entity. A task is similar to thread in most respects. A thread does not call another thread to run. A task also does not directly call another task to run. Both need an appropriate scheduler. Multithreading needs a thread scheduler and a task scheduler.

Scheduling Task & Threads.

The OS kernel has a scheduler for tasks or threads. A task can be considered to be in one of the five states.

(i) Idle: Tasks has been created and memory allotted to its structure.

(ii) Ready (Active) :- The created task is ready and

KTUNOTES.IN

Downloaded from Ktunotes.in

is schedulable ~~scheduled~~ by the kernel but has not begun execution as another higher priority task is scheduled to run and got the s/m resources at this instance.

Running State:

Executing the servicing codes and getting the s/m resources at this instance. It will run until it needs some i/ps or start wait for an event or till it pre-empts by another higher priority task than this.

Blocked (waiting)

Execution of the servicing codes suspend after saving needed parameters into its context. It needs some i/p (IPC), or waiting for an event, or waiting for higher priority task to block. Ex: a task is ~~blo~~ pending while it waits for an i/p from the keyboard or a file. The scheduler then puts it in the blocked state.

Deleted (finished)

A created task has memory de-allotted to its structure. It frees the memory. Task has to be re-created.

A created and activated task will be in one of three states, ready, running and blocked.

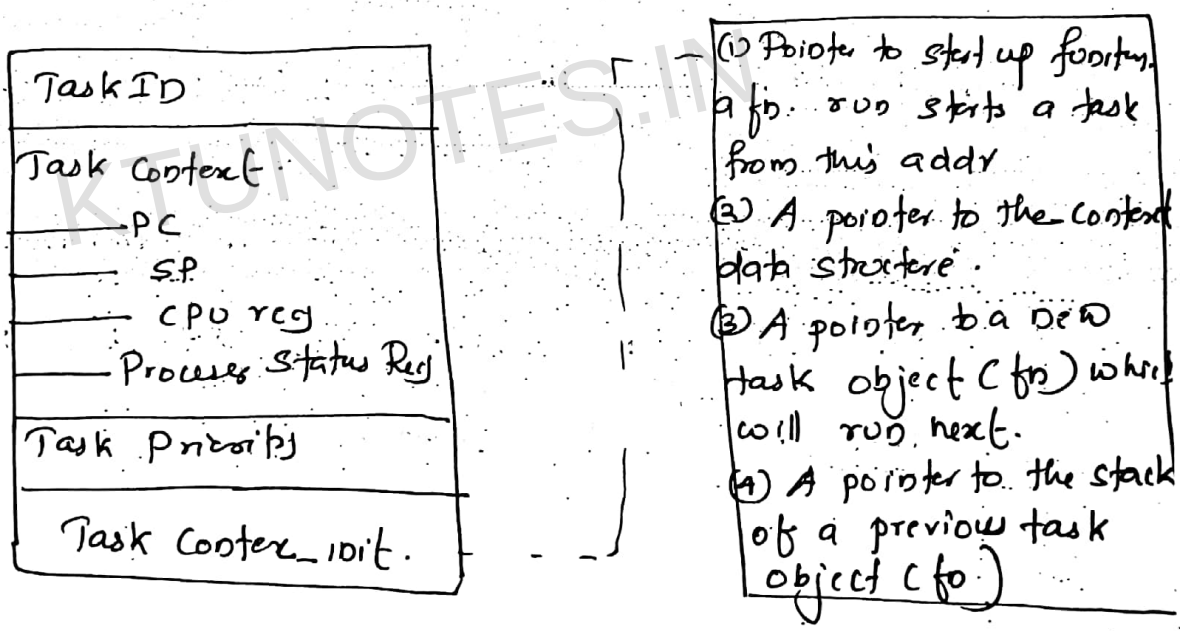
KTUNOTES.IN

Downloaded from Ktunotes.in

Pending Threads.

A pending thread is one that is waiting for OS scheduler to schedule or waiting for a signal or message. When there is one CPU and no of threads, only one task or thread runs and others are pending. Pending threads run according # to the priority in pre-emptive scheduling and according to allocated time in non pre-emptive scheduling.

Tasks and Data.



KTUNOTES.IN

Task and its data including context are TCB.

Task ID → Each task has an ID. May be a no. of 1 byte, called identifier of task (0-255)

Priority Parameter → 0-255. rep. by byte - higher the value, lower the priority.

Downloaded from Ktunotes.in

Each task has independent values of PC, SP. (Memory addr. from where it gets the saved cpu registers and parameters).

Task Control Block (TCB)

Each task consists of a TCB which is a memory block. The TCB is a data structure having the information using which the OS controls the task state. The TCB stores in protect area of kernel. Give TCB has the following information about the task.

- It stores the current program counter information. (addr. of next instru. to be executed)
- Memory map, the signal dispatch table, task ID, cpu state (reg, task PC, task SP) and a kernel stack.

KTUNOTES.IN

Downloaded from Ktunotes.in

Inter Process Communication:

Interprocess communication functions in a multiprocessor s/m are used

1. To interrupt present process for an interrupt service by another processes.
2. To notify occurance of an event to another process waiting for the event.
3. To set/reset a signal, token or flag to generate message from the certain set of computations finishing on one task, and let the other tasks take note of signal or get the msg.
4. To generate information abt certain set of computations finishing on one process, to let the other process wait for finishing those computations and when computations finish then take note of information and let others generate information.
5. To generate some information in a process or value or generate a message in output so that it lets another process take note or use it through the kernel.

Message may be a msg of known size, o/p data or header words plus data. One way is use of global variable, data structures/objects

KTUNOTES.IN

Downloaded from Ktunotes.in

IPC means that functions for the following

1. Signals.
2. Semaphores.
3. Message Passing
4. Message queue
5. Mailboxes
6. Pipes.
7. Sockets in TCP/IP & Unix
8. File and Memory mapped file.
9. Shared Memory.

Signals:

One way for messaging is to use an OS function signal(). It is provided in Unix, Linux and several OSs such as Vx works. Unix uses signals and has 64 different type of signals for the various actions / events. An OS has provision for issuing signals.

The process sending the signal uses C- and an integer no. in the argument. Signal handler (ISR) name is mostly implicit to OS from signal no. or if not implicit then specified by the user.

KTUNOTES.IN

Downloaded from Ktunotes.in

following are the ways of issuing a signal

1. OS itself can issue an interrupt signal.
2. Process (task/thread) on executing function signal( ) sends an interrupt signal to the OS. Signal issuing means execution of a s/w-interrupt instruction. A signal is assigned a signal no. by OS.

Ex: SIGFPE is assigned no. 8.

The instn signal (8) issues when an illegal mathematical operation is attempted.

The signal() forces a process/task called signal handler to run. Signal handler may be a process/task and has coding similar to ones in an ISR. The handler runs in a way similar to highest priority ISR. Unless masked by a signal mask, the signal allows the execution of the signal-handling function and allows the handler to run just as a hardware interrupt allows the execution of an ISR.

An advantage of signal is that it takes shortest possible CPU time to force a handler to run. Signals may have default actions; stop a process, continue a stopped process, dump the core in a file, ignore the signal or terminate a process

KTUNOTES.IN

Downloaded from Ktunotes.in

An important appln. of signal is to handle exceptions: [Exception is a process that is executed on a specific reported run time condn.]. A signal reports an error during the running of a task and then let the scheduler initiate an error handling process.

Concept of Semaphores:

KTUNOTES.IN

Downloaded from Ktunotes.in

Inter Process Communication

A process can be of two type:

- Independent process.
- Co-operating process.

An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently but in practical, there are many situations when co-operative nature can be utilised for increasing computational speed, convenience and modularity. Inter process communication (IPC) is a mechanism which allows processes to communicate each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other using these two ways:

1. Shared Memory
2. Message passing

The Figure 1 below shows a basic structure of communication between processes via shared memory method and via message passing.

An operating system can implement both method of communication. First, we will discuss the shared memory method of communication and then message passing. Communication between processes using shared memory requires processes to share ome variable and it completely depends on how programmer will implement it. One way of communication using shared memory can be imagined like this: Suppose process1 and process2 are executing simultaneously and they share some resources or use some information from other process, process1 generate information about certain computations or resources being used and keeps it as a record in shared memory. When process2 need to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly. Processes can use shared memory for extracting information as a record from other process as well as for delivering any specific information to other process. Let's discuss an example of communication between processes using shared memory method.

KTUNOTES.IN

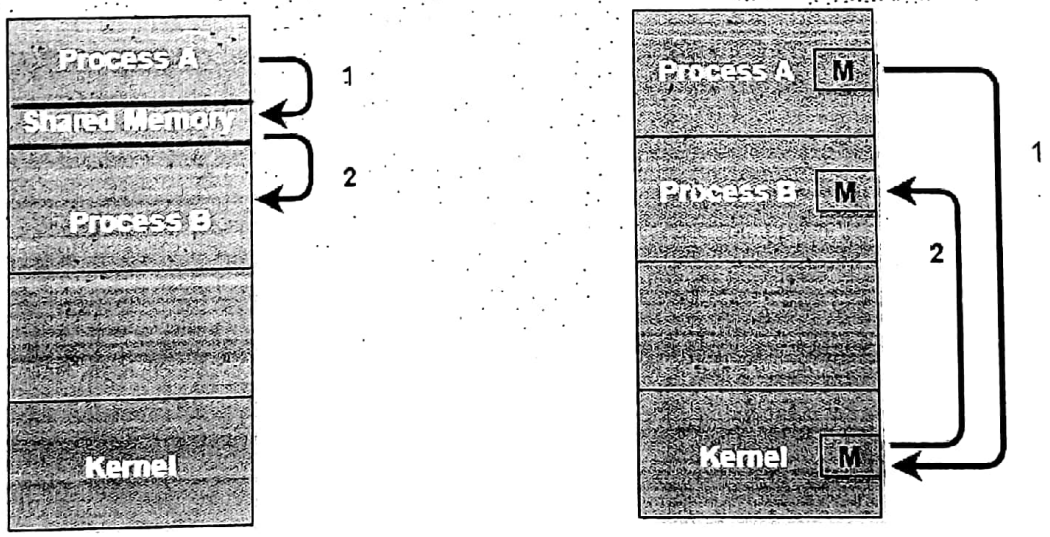


Figure 1 - Shared Memory and Message Passing

Downloaded from Ktunotes.in

Semaphores

A semaphore, in its most basic form, is a protected integer variable that can facilitate and restrict access to shared resources in a multi-processing environment. The two most common kinds of semaphores are counting semaphores and binary semaphores. Counting semaphores represent multiple resources, while binary semaphores, as the name implies, represents two possible states (generally 0 or 1; locked or unlocked). Semaphores were invented by the late Edsger Dijkstra.

Semaphores can be looked at as a representation of a limited number of resources, like seating capacity at a restaurant. If a restaurant has a capacity of 50 people and nobody is there, the semaphore would be initialized to 50. As each person arrives at the restaurant, they cause the seating capacity to decrease, so the semaphore in turn is decremented. When the maximum capacity is reached, the semaphore will be at zero, and nobody else will be able to enter the restaurant. Instead the hopeful restaurant goers must wait until someone is done with the resource, or in this analogy, done eating. When a patron leaves, the semaphore is incremented and the resource becomes available again.

A semaphore can only be accessed using the following operations: wait() and signal(). wait() is called when a process wants access to a resource. This would be equivalent to the arriving customer trying to get an open table. If there is an open table, or the semaphore is greater than zero, then he can take that resource and sit at the table. If there is no open table and the semaphore is zero, that process must wait until it becomes available. signal() is called when a process is done using a resource, or when the patron is finished with his meal. The following is an implementation of this counting semaphore (where the value can be greater than 1):

KTUNOTES.IN

Historically, wait() was called P (for Dutch "Proberen" meaning *to try*) and signal() was called V (for Dutch "Verhogen" meaning *to increment*). The standard Java library instead uses the name "acquire" for P and "release" for V.

No other process can access the semaphore when P or V are executing. This is implemented with atomic hardware and code. An atomic operation is indivisible, that is, it can be considered to execute as a unit.

If there is only one count of a resource, a binary semaphore is used which can only have the values of 0 or 1. They are often used as mutex locks. Here is an implementation of mutual-exclusion using binary semaphores:

In this implementation, a process wanting to enter its critical section it has to acquire the binary semaphore which will then give it mutual exclusion until it signals that it is done.

For example, we have semaphore s, and two processes, P1 and P2 that want to enter their critical sections at the same time. P1 first calls wait(s). The value of s is decremented to 0 and P1 enters its critical section. While P1 is in its critical section, P2 calls wait(s), but because the value of s is zero, it must wait until P1 finishes its critical section and executes signal(s). When P1 calls signal, the value of s is incremented to 1, and P2 can then proceed to execute in its critical section (after decrementing the semaphore again). Mutual exclusion is achieved because only one process can be in its critical section at any time.

Downloaded from Ktunotes.in

Disadvantage

As shown in the examples above, processes waiting on a semaphore must constantly check to see if the semaphore is not zero. This continual looping is clearly a problem in a real multiprogramming system (where often a single CPU is shared among multiple processes). This is called busy waiting and it wastes CPU cycles. When a semaphore does this, it is called a spinlock.

To avoid busy waiting, a semaphore may use an associated queue of processes that are waiting on the semaphore, allowing the semaphore to block the process and then wakeit when the semaphore is incremented. The operating system may provide the block() system call, which suspends the process that calls it, and the wakeup(P <Process>)system call which resumes the execution of blocked process P. If a process calls wait() on a semaphore with a value of zero, the process is added to the semaphore's queue and then blocked. The state of the process is switched to the waiting state, and control is transferred to the CPU scheduler, which selects another process to execute. When another process increments the semaphore by calling signal() and there are tasks on the queue, one is taken off of it and resumed.

In a slightly modified implementation, it would be possible for a semaphore's value to be less than zero. When a process executes wait(), the semaphore count is automatically decremented. The magnitude of the negative value would determine how many processes were waiting on the semaphore.

KTUNOTES.IN
RIET

Downloaded from Ktunotes.in