1 Chapter 4: Intertask Communication, Synchronization and Memory management.

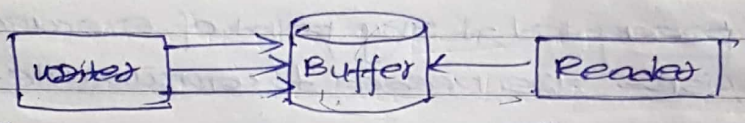
- RTOS offers a variety of mechanisms for communication and synchronization between the task to prevent corrupted communication and interference between them.
- They provide several mechanisms with each mechanism optimized for reliably passing a different kind of information from task to task where all tasks are not independent and they can't be preempted at any point of execution.
- These mechanisms are needed to communicate, share resource and synchronize activity.
- It is used for minimizing blocking that may arise in a uniprocessor system when concurrent tasks use shared resources.
- Task synchronization is acquired to maintain the consistency / integrity of the shared data / resources that can only be used by one task at a time.

4.1.1 Buffering Data: Time Relative Buffering, Ring Buffers (use गर्ने होइन)

- Global variables can be used to pass data between tasks in a multitasking system But tasks of higher priority can preempt lower-priority routines at inappropriate times, corrupting the global data.
- one task may produce data at a constant 100 units per second where as another may consume these data at rate less than 100 unit per second.
- This problem can be solved if the produces fills a storage buffer with the data. The buffer holds the excess data until the consumer task can catch up.

2.

- A buffer is a region of a physical memory storage used to temporarily store data while it is being moved from one place to another.
- A buffer often adjusts timing by implementing a queue in memory by writing data into the queue at one rate and reading it at another rate



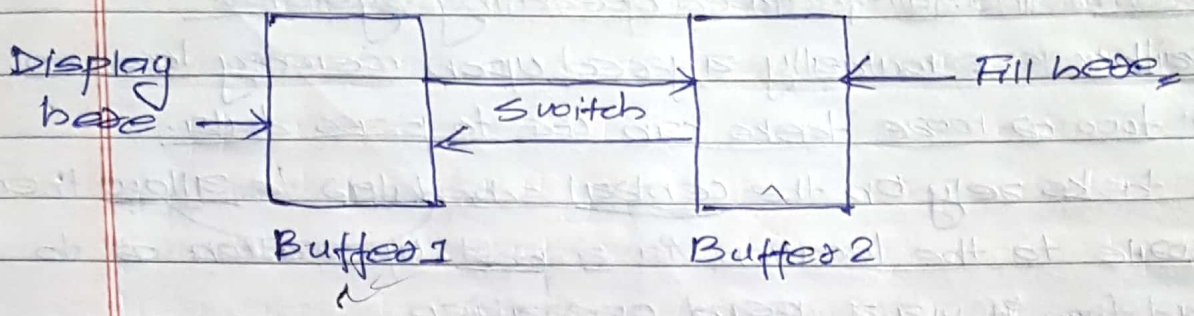
Time Relative Buffering:-
- This technique is used when time relative (synchronous) data need to be transferred between cycles of different rates or when a full set of data is needed by one process but can only be supplied slowly by another process
- Two buffers are used in this technique, so it is also known as double buffering or ping pong buffering.
- Hardware or software switch is used to alternate the buffers.
- It is used in disk controllers, graphical interfaces, navigation equipment, robot controls etc.

Example:
In graphics processes, lines are drawn one by one until the image is completed. In an animation system, we don't want to see this drawing process. In this case, time relative buffering can be used. We draw on one screen when the new drawing is complete so that the line

Howe T.B helps in interprocess communication & synchronization? Justify. 7.

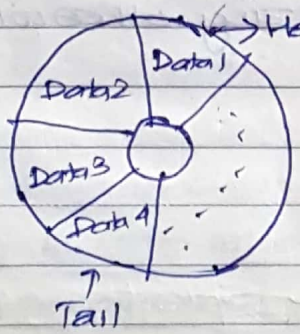
3.

drawing commands will not be seen.



Imp # Ring Buffers:- (Circular queue Use गरेर बनाइने Buffer).

- It is a type of buffer that is formed by using a special data structure called a circular queue (FIFO)
- Simultaneous input and output to the list are achieved by keeping head and tail pointers.
- Data are added at the tail and read from the head.



Pseudocode

▷ Let Q be the queue, n be the size of queue array, head be the array index where the data is read and tail be the array index where the data is filled. Initially when data list full, head = tail = 0

Dequeue:

▷ If the queue is empty then print underflow. else, store the data at index 'head' into a variable and head = (head+1) MOD n

Enqueue:-

▷ If the queue is full then print overflow else, insert data in the tail and tail = (tail+1) MOD n

4.

4.1.2 Mailboxes (message exchanges)

- It is an intertask communication device available in many commercial and full-featured operating systems.
- A mailbox is a mutually agreed upon memory location that two or more tasks can use to pass data.
- The tasks rely on the central scheduler to allow them to write to the location via a post operation or to read from it via a pend operation.
- The difference between the pend operation and simply polling the mailbox is that the pending task is suspended while waiting for data to appear. Thus, no time is wasted continually checking the mailbox, ie the busy waiting condition is eliminated.

```
pend (data, S) {
  data of mailbox S is returned if any, otherwise
  task is suspended
}

void post (data, S) {
  data is kept on the mailbox S
}
```

5.

4.1.3 Critical Regions:-

- Code that interact with a serially reusable resource is called CR.
- In multitasking/ multiprocessing system, serially reusable resource is called if are those shared resources that can only be used by one task at a time, and use of resource cannot be interrupted.
- If two tasks enter the same critical section/region simultaneously a catastropic error can occur.

Example,

Task A prints "I am Task A".

Task B prints "I am Task B"

Let us assume, Task B was printing and was interrupted by Task A. The result would be:

I am I am Task A Task B.

4.1.4 Semaphores:

- It is an abstract data type (ADT)
- It involves a special variable called a semaphore and two operations on that variable (namely "wait" and "Signal").
- Wait operation is used to set the semaphore and denoted by P(S).
- Signal operation is used to reset the semaphore and denoted by V(S).

6. void wait (boolean s)

```
void P(boolean S){
while(S==TRUE);
   S=TRUE;
}
```

```
void V(boolean S){
   S=FALSE;
}
```

wait → P(S) → set semaphore & enter CR
Signal → V(S) → exit C.R & reset semaphore.

```
Procedure process1;
   :
P(S)   // wait(S)
// Critical Region
) write("I am Task A");
   V(S)
   end.
```

```
Procedure process2
   :
P(S)
// critical Region
write("I am Task B");
V(S)
:
end.
```

begin P(S) // Critical Sect write (...) V(S) end

Mailboxes and Semaphores:-

- Mailboxes can be used to implement semaphores if semaphore primitives are not provided by the operating system.
- In this case, there is the added advantage that the pend instruction suspend the waiting process rather than actually waiting on the semaphore.

```
void P(boolean S){
   boolean KEY;
   pend(KEY, S);
}
```

```
void V(boolean s){
   post(KEY, S); /* Place key in
                    mailbox */
}
```

7.

Counting Semaphores:-

- A semaphore that can take on two or more values
- It can be used to protect pools of many resources, or to keep track of the number of free resources.
- The semaphore must be initiated to the total number of free resources before real-time processing can commence

Wait operation:

```
void MP(int S){
    S = S-1;
    while (S<0);
}
```

Signal operation:

```
void MV(int S){
    S = S+1;
}
```

- Problem may arise in semaphore if the operation of testing and subsequently setting a semaphore is not atomic

```
void P(boolean S){
  while(S == TRUE);
    S = TRUE;
}
```

Task 1 and Task 2 are running in round robin fashion. When T1 completes line 2, let the quantum time for T1 end. Then T2 execution begins. It also sees S as false. So, it will enter critical section by making S -> true and enters critical section. So, there will be two processes at the CR.

8.

V. IMP

4.1.5 Resource Contentions and Priority Inversion.

Resource Contention :-

- It is a conflict over access to a shared resource such as random access memory, disk storage, cache memory, internal buses or external network devices.
- Resolving resource contention problems is one of the basic functions of operating systems. Various low level mechanisms can be used to aid this, including locks, semaphores, mutexes and queues.
- Failure to properly resolve resource contention problems may result in number of problems including deadlock, livelock and thrashing.

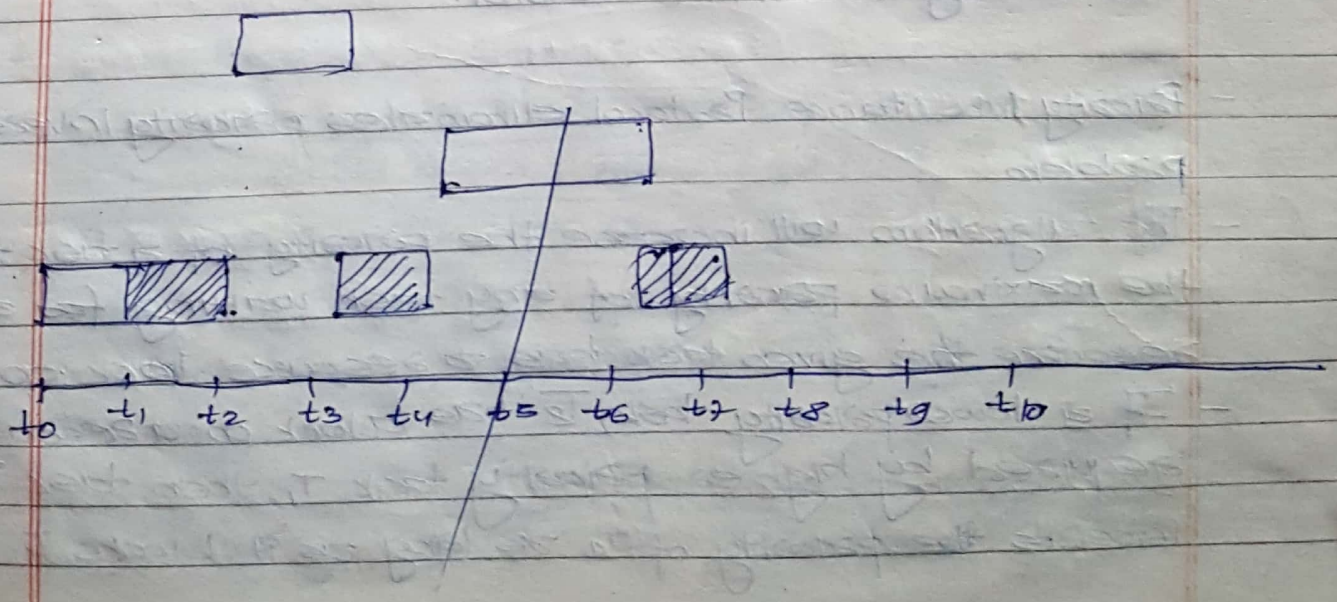
Priority Inversion :-

- It is a problematic scenario in scheduling in which a high priority task is indirectly preempted by a medium priority task. effectively "inverting" the relative priorities of the two tasks.
- Priority inversion is an undesirable situation in which a higher priority task get blocked (waits for CPU). for more time than that it is supposed to, by lower priority tasks.

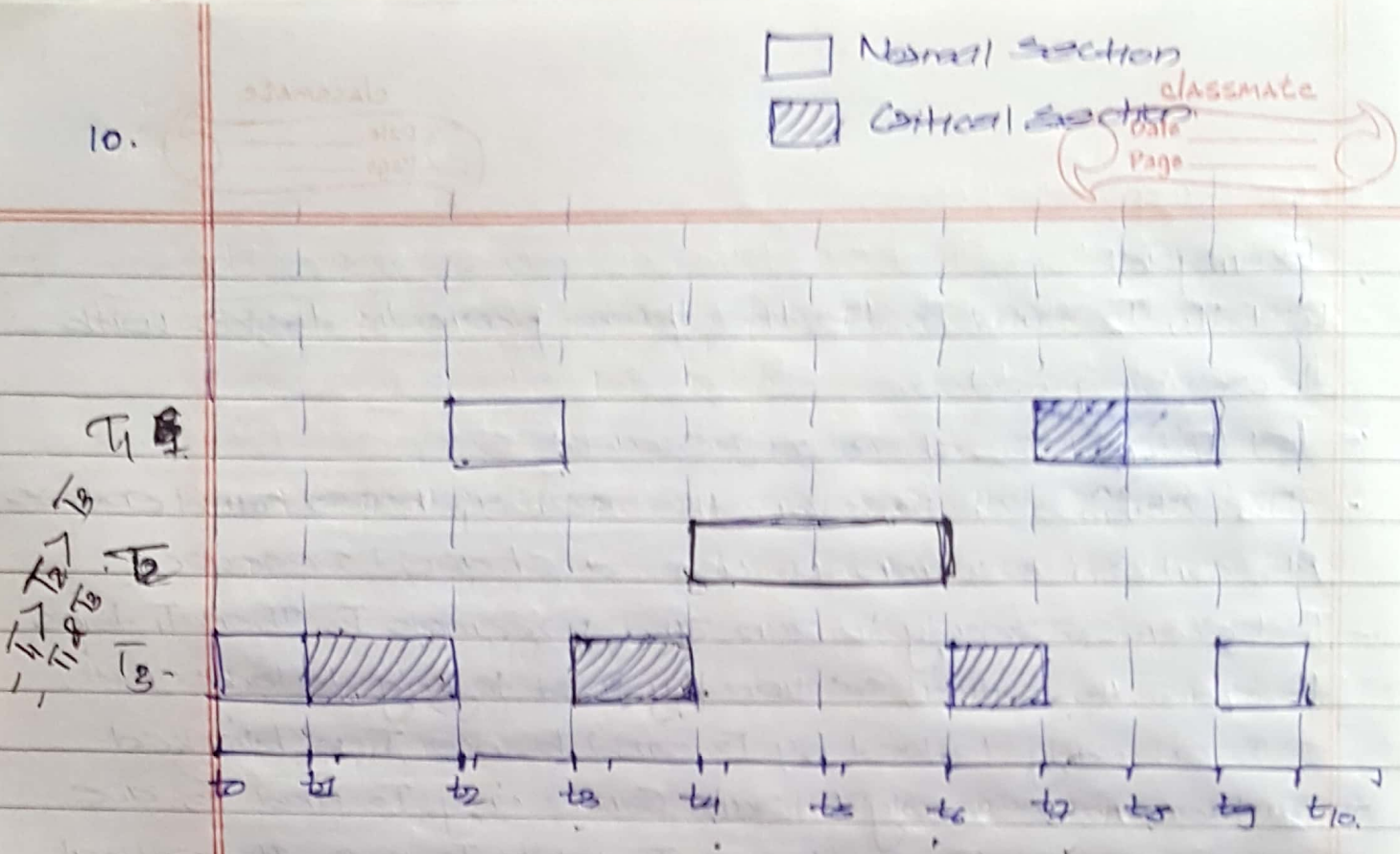
9.

Example:

- Let T_1, T_2 and T_3 be the three periodic tasks with decreasing order.
- Let T_1 and T_3 share a resource S.
- T_3 obtains a lock on the semaphores and enters its critical section to use a shared resource.
- T_1 becomes ready to run and preempts T_3. Then T_1 tries to enter its critical section by first trying to locks. But, S is already locked by T_3 and hence T_1 is blocked.
- T_2 becomes ready to run. Since only T_2 and T_3 are ready to run, T_2 preempts T_3 while T_3 is in its critical section.
- The execution time of T_2 increases the blocking time of T_1.
- Task T_2 completes its execution.
- T_3 completes its execution.
- At last T_1 can completes its execution



10.



Priority inversion is said to have occurred within the time interval $[t_4, t_6]$ during which the highest priority task T_1 has been prevented from execution by a medium priority task T_2.

Methods for Solving Priority Inversion Problem.

4.1.5.1 The Priority Inheritance Protocol.

- Priority Inheritance Protocol eliminates priority inversion problem.
- The algorithm will increase the priority of a task to the maximum priority of any task waiting for any resource the given task has a resource lock on.
- If a lower priority task, T_2 has a lock on a resource required by higher priority task T_1, then task T_2 inherits the priority of T_1 as long as it blocks T_1.

11.

When T_2 exits the critical section that caused the block, it reverts to the priority it had when it entered that critical section.

- Priority inheritance is transitive. i.e If T_3 blocks T_2, which blocks T_1, then T_3 inherits the priority of T_1 via T_2.
- It doesn't prevent the occurance of deadlocks.

4.1.5.2 Priority ceiling Protocol:-

- In this protocol, each resource is assigned a priority ceiling, which is the priority equal to the highest priority of any task which may lock the resource.
- The protocol works by temporarily raising the priorities of tasks in certain situation.
- If a high priority task is blocked through a resource, then the task holding that resource gets the priority of the high priority task. Once the resource is released the priority is reset back to its orginal.

4) Event Flag and Signals:-

- Synchronization mechanism is provided by certain languages called event flags. (कुनै event occur हुँदा flag लाई set गर्ने)
- It allows specification of an event that causes the setting of some flag. (specify गर्न सकिन्छ)
- Even Flag represents simulated interrupts, created by the programmer.
- Raising the event flag transfers flow-control of control to the operating system, which can then invoke the appropriate.

handler.

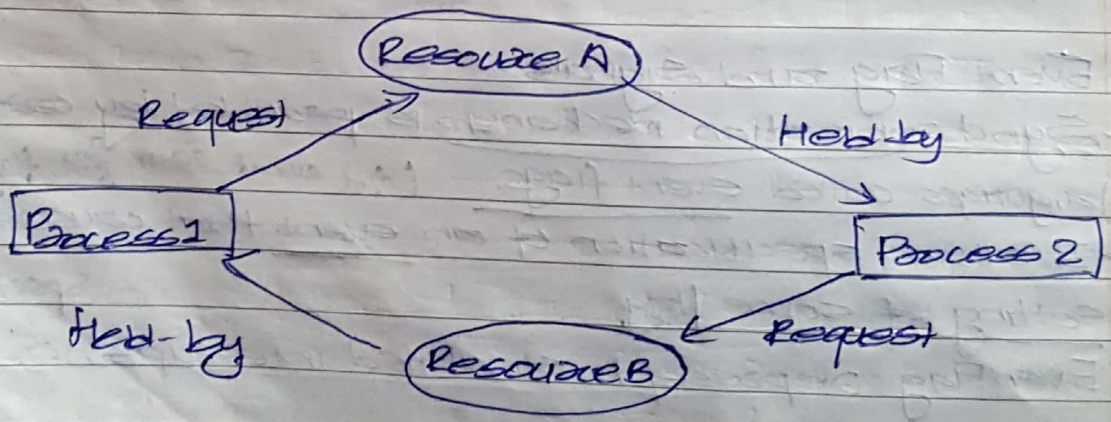
- Tasks that are waiting for the occurance of an event are said to be blocked.

- A signal is a type of software interrupt handler that is used to react to an exception indicated by the raise operation.

Deadlock:-

- Deadlock is a situation that occurs when multiple process are waiting for the availability of a resource that will not become available because it is being held by another process that is in a similar wait state.

- A situation that occurs when multiple processes are waiting for an action by or response from another process that is in a similar wait state.



- Deadlock is serious problem because it cannot always be detected through testing. It may occur

13.

very infrequently, making the pursuit of deadlock problem more difficult.

Example:-

Task A and Task B require resources 1 and 2. Task A is in possession of resource 1 but waiting on resource 2. Task B is in possession of resource 2 but waiting on resource 1. ~~Task B is in possession of resource 2~~ Neither Task A nor Task B will release the resource until its other request is satisfied.

Starvation: (aka Livelock)

A condition in which a process is indefinitely delayed because other processes are always given preference. Starvation differs from deadlock in that at least one process is satisfying its requirements but one or more are not. In deadlock, no process can satisfy their requirements because all are blocked.

Four conditions are necessary for deadlocks.

1) Mutual Exclusion:

At least one resource must be non-shareable. Only one process can use the resource at any given instant of time.

14.

iii) Circular Wait:-
- A process must be waiting for a resource which is being held by another process, which is intern waiting for the first process to release the resource.

iii) Hold and Wait:-
- A process is currently holding atleast one resource and requesting additional resources which are being held by other processes.

iv) No preemption:-
- The operating system must not de-allocate resources once they have been allocated; they must be released by the holding process itself.

Deadlock Avoidance:-
- The best way to deal with deadlock is to avoid it altogether.
- Better to have a dynamic technique that examines each new resource request for deadlock. If the new request could lead to a deadlock; then the request is denied.

Recommendations to avoid deadlocks:-
- Minimize the number of critical region and their size.
- All processes must release any locks before returning to the calling function
- Do not suspend any task while it controls a critical region.

15.

- All critical regions must be made free.
- Do not lock devices in interrupt handlers.
- Always perform validity checks on pointers used within critical region.

4.2 Real Time Memory Management

4.2.1 Process Stack Management:-

- In multitasking system, context for each task needs to be saved and restored in order to switch processes.
- We can use Task Control Block (TCB) model for saving and restoring context in full featured real time operating system.
- We can use RunTime stacks for saving and restoring context in interrupt only system and foreground/background systems.

4.2.3 Memory Management in the Task-Control-Block Model.

- In task-control-block model a (fixed or dynamic) list of task control blocks is kept.

list

- In the fixed case, n task-control-blocks are allocated at system generation time in the inactive state of that task
- As tasks are created, the task-control-block enters the ready state.
- Prioritization or time slicing will then move the task to the execute state.

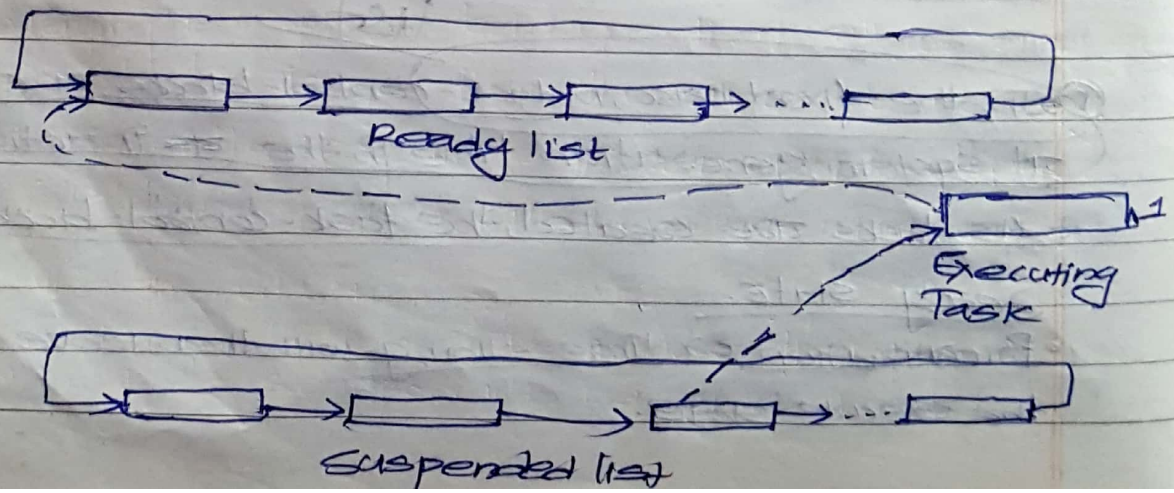
- If a task is to be deleted, its task-control-block is simply placed in the deleted state.
- No real time memory management is necessary in case of fixed number of task-control-blocks.

linked list

- In the dynamic case, task-control-blocks are added to a linked list or some other dynamic data structure as tasks are created.
- Tasks are in the suspended state upon creation and enter the ready state via an operating system call or event.
- Prioritization or time slicing will then move the task to the executing state.
- When a task is deleted, its task-control block is removed from the linked list, and its heap memory allocation is returned to the available status.
- Real time memory management is required and consists of managing the heap and other data structures such as list or queue. (special type of data structure based on binary tree).

System → Ready List
→ Suspended List
Table → Resource
→ Resource Request.

Fig: Memory management in real time kernel.



17.

routine - save
. . resove.

b) Run-time stack:-

- If a runtime stack is to be used for saving the context, then two routines are required "save" and "restore".
- "save" is called by interrupt handler to save the current context to stack, immediately after interrupts have been disabled.
- "restore" is called by interrupt handler just before interrupts are enabled and before returning from the interrupt handler.

- A run-time stack cannot be used in a round-robin system because of the first-in/first-out nature of the scheduling. In this case ring buffer or circular queue can be used to save context where context is saved to the tail of the list and restored from the head.

4.2.2. Multiple-Stack Arrangement:-

- Often a single run-time stack is inadequate to manage several processes in foreground/background system.
- So, a multiple stack scheme uses a single run time stack and several application stacks.

Advantages:-

- It permits tasks to interrupt themselves, thus allowing for detection of false interrupts and ~~handling~~ handling overload conditions.
- supports recursion, such as in C or Pascal.

classmate
Date
Page

Swapping :-

- It is the simplest scheme that allows an operating system to allocate memory to two tasks "simultaneously".
- In this case, the operating system is always memory resident, and one task can co-reside in the memory space not acquired by the operating system, called the user space.
- When a second task needs to run, the first task is suspended and then swapped, along with its context, to a secondary storage device, usually hard disk.
- The second task, along with its context, is then loaded into the user space and initiated by the task dispatcher.
- This type of memory management scheme can be used along with round robin or preemptive priority systems, and is highly desirable to have the execution time of each task be long relative to the lengthy memory - disk - memory swap delay.

Overlays

→ Overlaying is a general technique that allows a single program to be larger than the allowable memory.
→ In this case, the program is broken up into dependent code and data sections called overlays, which can fit into the available memory space.
→ Special program code must be included that permits new overlays to be swapped into memory as needed

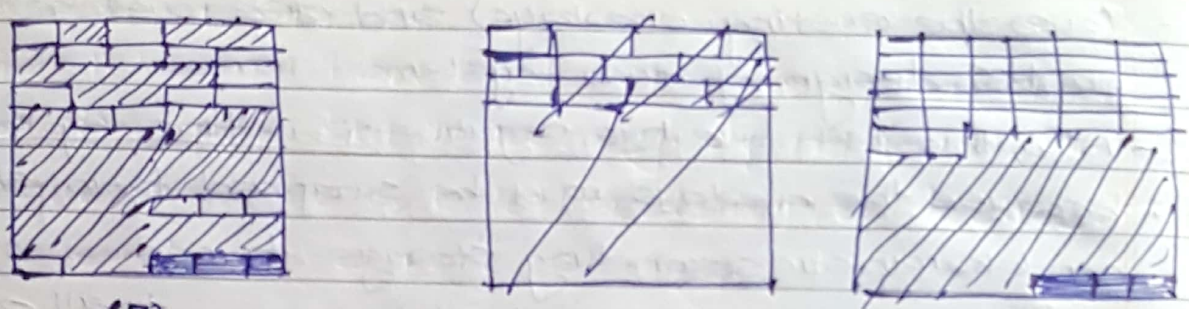
(uses the existing overlays) and care must be exercised in the design of such systems.

- Also, this technique has negative real-time implications, because the overlays must be swapped from slow and nondeterministic secondary storage devices.
- In overlays, any code that is common to all overlays is called root.
- In Both swapping and overlays, one portion of memory is never swapped or overlaid. This critical memory segment contains the swap or overlay manager.

4.2.6 Block or Page Management

Block Management:-

- To allow more than one process to be memory-resident simultaneously, user space can be divided into fixed-size partitions called block.
- This scheme is useful in systems where the number of tasks to be executed is fixed.
- Partition swapping to disk can occur when a task is preempted.
- Tasks must reside in continuous partitions, and the dynamic allocation and deallocation of memory can cause problems.



fig; Fragmented memory (a) before and (b) after compaction, including unmovable blocks

- In fixed block management, ~~two blocks~~ both internal and external ~~block management~~ fragmentation occur.

- Internal fragmentation occurs when a process requires less memory than available block size.
- The internal fragmentation can be reduced by creating fixed partitions of several sizes and then allocating the smallest partition greater than the required amount.

- Both internal and external fragmentation harmper efficient memory usage and degrade real-time performance due to associated overhead to bt a process to available memory and disk swapping.

→ To remove the internal fragmentation, ~~variable size~~ memory is allocated in amounts that are determined by the requirements of the process to be loaded into memory or compaction is used.
Variable length blocks allocation.
→ ~~This~~ technique is ~~more~~ appropriate when the number of real-time tasks is unknown or varies.
→ External fragmentation may still occur because of the dynamic nature of memory allocation and deallocation because memory must still be allocated to a process contiguously.

→ Compressing fragmented memory or compaction can be used to mitigate internal fragmentation.
→ Compaction is a CPU intensive process and not encouraged in hard-real time systems.
→ If compaction must be performed, it should be done in the background, with interrupts disabled.

→ To remove the external fragmentation, Demand page system is used.
→ In demand page systems, program segments are permitted to be loaded in ~~main memory~~ non contiguous memory. ~~as they are requested by~~
→ Paging ~~is~~ allows non consequtive references to pages via a page table.
→ Pointers are used to access the desired page
→ When a memory reference is made to a location within a page not loaded in main memory, a page fault

Q2.

exception is raised.

- The interrupt handler for this exception checks for free page slot in memory. If not found, a page frame must be selected and swapped to disk. This process is called page stealing.

- Paging can lead to problems like thrashing (high paging activity), internal fragmentation and deadlock.
- Paging is not usually used in real time systems because, it has high overhead for mapping & hardware support is not usually not available)