CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Internal Assesment Test – III

Sub:	**EMBEDDED SYSTEM DESIGN/ ARM MICROCONTROLLER & EMBEDDED SYSTEM**		Sec	A			Code:	18EC62
Date:	29/ 07 / 2021	Duration:	90 mins	Max Marks:	50	Sem: VI	Branch:	ECE /TCE

Note: Answer any five full questions.

		Marks	OBE	
			CO	RBT
1	Explain the assembly language based embedded firmware development with a diagram and mention its advantages and disadvantages.	10	C604.1	L1
2	Explain the different Embedded firmware design approaches in detail.	10	C604.1	L2
3	Write a note on Monolithic kernel and Micro kernel?	10	C605.1	L2
4	Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds respectively enters the ready queue together. A new process **P4 with estimated completion time 2 ms enters the 'Ready' queue after 2 ms.** Assume all the processes contain only CPU operation and no I/O operations are involved. Calculate the waiting time and turn around time of individual process and also Average waiting time and turnaround time in Preemptive SJF Scheduling/Shortest Remaining Time (SRT) Algorithm .	10	C605.1	L1
5	Explain out of circuit and in-system programming methods for integration of hardware and firmware.	10	C605.1	L1
6	Write note on counting semaphore and Binary semaphore	10	C605.1	L1

3) Monolithic Kernel

- In monolithic kernel architecture, all kernel services run in the kernel space.
- Here all kernel modules run within the same memory space under a single kernel thread.
- The tight internal integration of kernel modules in monolithic kernel architecture allows the effective utilization of the low-level features of the underlying system.
- The major drawback of monolithic kernel is that any error or failure in any one of the kernel modules leads to the crashing of the entire kernel application.
- Examples of monolithic kernel are LINUX, SOLARIS, MS-DOS.
- The architecture representation is given below.

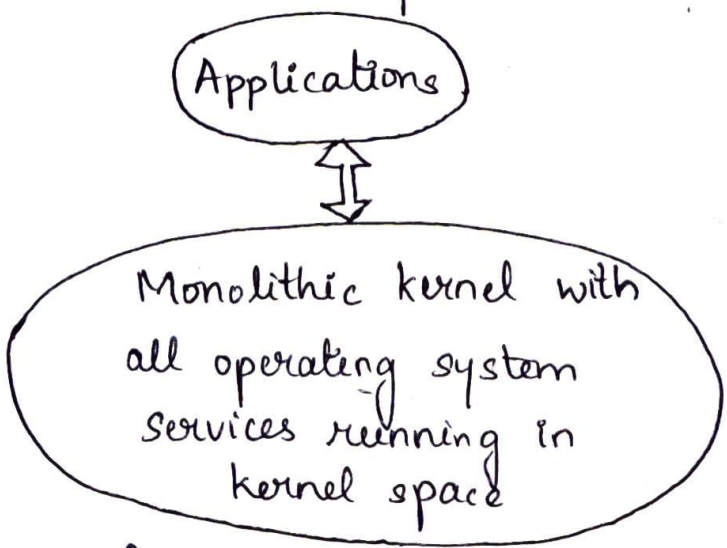


fig. The Monolithic Kernel Model

Microkernel

- The microkernel design incorporates only the essential set of operating system services into the Kernel.
- The rest of the Operating System services are implemented in programs known as 'Servers' which runs in user space.
- This provides a highly modular design and OS-neutral abstraction to Kernel.
- Memory management, process management, timer systems and interrupt handlers are the essential services, which forms the part of the microkernel.
- Examples are Mach, QNX, Minix 3.

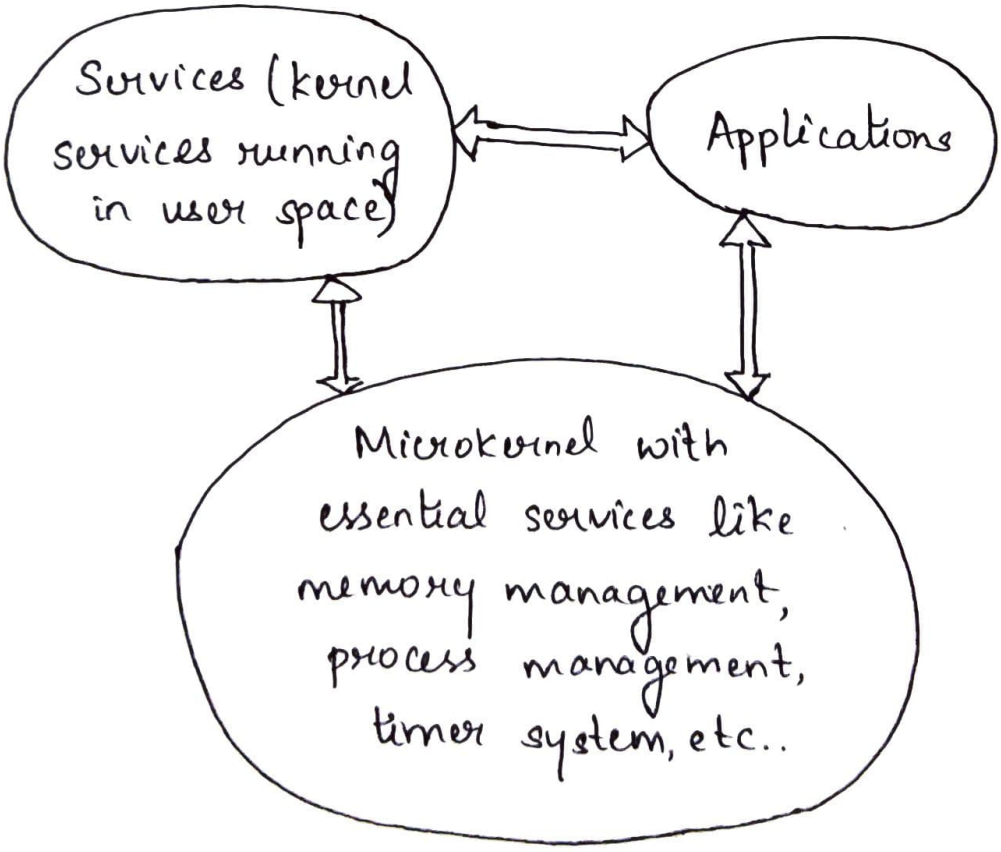


fig. The Microkernel model.

Microkernel has the following benefits:

→ Robustness: If a problem is encountered in any of the services, which runs as 'server' application, the same can be configured and re-started without the need for re-starting the entire OS. Thus it is useful for systems which demand high 'availability'.

→ Configurability: Since there is no need to restart the whole system, this makes the system dynamically configurable.

2) The firmware design approaches for embedded product is purely dependent on the complexity of the functions to be performed, the speed of the operation required, etc.

i) The Super Loop Based Approach

→ This is adopted for applications that are not time critical and where the response time is not so important.

→ It is similar to a conventional procedural programming where the code is executed task by task.

→ The task listed at the top of the program code is executed first and the tasks just below the top are

executed after completing the first task.

→ In a multiple task based system, each task is executed in serial in this approach.

→ The firmware execution flow for this will be

1. Configure the common parameters and perform initialisation for various hardware components memory, registers, etc.
2. Start the first task and execute it.
3. Execute the second task.
4. Execute the next task.
5. Execute the next task.
6. Execute the next task.
7. Execute the last defined task.
8. Jump back to the first task and follow the same flow.

→ The 'Super loop based design' does not require an operating system, since there is no need for scheduling which task is to be executed and assigning priority to each task.

→ The priorities are fixed, hence the task to be executed are also fixed.

→ A typical example of a 'Super loop based' product is an electronic video game toy containing keypad and display unit.

→ Even if the application misses a key press, it will only come as a bug in firmware and won't create issues.

→ Major drawback of this approach is the lack of real timeliness.

→ If the number of tasks to be executed within an application increases, the time at which each task is repeated also increases.

ii) The Embedded Operating System (OS) Based Approach

→ The OS based approach contains operating systems, which can be either a General Purpose Operating System (GPOS) or a Real Time Operating System (RTOS) to host the user written application firmware.

→ The GPOS based design is similar to a conventional PC based application development where the device contains an OS and you will be creating and running user applications on top of it. Example: Microsoft Windows XP Embedded.

→ For developing applications on top of OS, the OS supported APIs are used.

→ Similar to the different hardware specific drivers, OS based applications also require 'Driver software' for different hardware present on the board to communicate with them.

→ Real Time Operating System (RTOS) based design approach is employed in embedded products demanding real time response.

→ RTOS responds in a timely and predictable manner

to events.

→ RTOS contains a Real Time kernel responsible for performing pre-emptive multitasking, scheduler for scheduling tasks, multiple threads, etc.

→ It allows flexible scheduling of system resources like the CPU and memory and offers some way to communicate between tasks.

→ Examples of RTOS are 'Windows CE', 'pSOS', 'VxWorks' 'ThreadX', etc.

1) Assembly language programming is the task of writing processor specific machine code in mnemonic form, converting the mnemonics into actual processor instruction (machine language) and associated data using an assembler.

Assembly language program was the most common type of programming adopted in the beginning of software evolution if we look back to the history of programming we can use that a large number of programs were written in assembly language including most popular game written for the super Nintendo Entertainment system the popular game NBA In 1993 was also coded entirely using the assembly language.

Advantages

→ It allows complex jobs to run in a simpler way.

→ It is memory efficient. as it requires less memory.

→ It is faster in speed, as its execution time is less.

→ It is mainly hardware oriented.

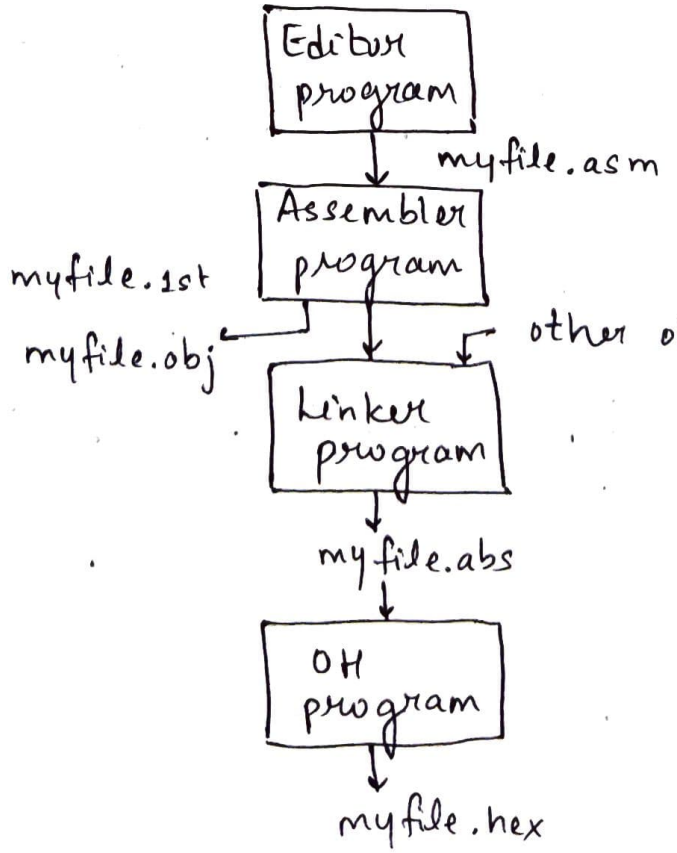
→ It is a low level embedded system.

Disadvantages

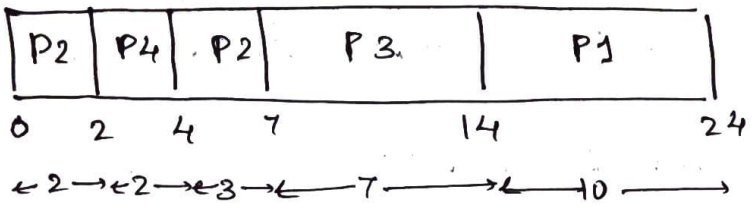
→ It takes a lot of time and effort to write the code for the same.

→ It is complex and difficult to understand.

→ The syntax is difficult to remember.



4) At the beginning, there are only three processes (P1, P2 and P3) available in the 'Ready' queue and the SRT scheduler picks up the process with the shortest remaining time for execution completion for scheduling. Now process P4 with estimated execution completion time 2ms enters the 'Ready' queue after 2ms of start of executing of P2. The processes are re-scheduled for execution in the following order.



The waiting time for all the processes are given as

Waiting time for P2 = 0ms + (4-2) ms = 2ms

Waiting time for P4 = 0ms

Waiting time for P3 = 7ms

Waiting time for P1 = 14ms

Average waiting time = $\frac{\text{Waiting time for all the processes}}{\text{No. of Processes}}$

$= \frac{\text{Waiting time for (P4+P2+P3+P1)}}{4}$

$= \frac{(0+2+7+14)}{4}$

$= 5.75$ milliseconds

Turn Around Time (TAT) for P2 = 7ms

(Time spent in Ready Queue + Execution time)

Turn Around Time (TAT) for P4 = 2ms

(Time spent in Ready Queue + Execution Time
= Execution Start Time − Arrival Time) +
Estimated Execution time = (2 − 2) + 2)

Turn Around Time (TAT) for P3 = 14ms

Turn Around Time (TAT) for P1 = 24ms

Average Turn Around Time = (Turn Around Time for all the processes) / No. of processes

= ~~Time~~

= Turn Around Time for (P2+P4+P3+P1) / 4

= (7 + 2 + 14 + 24) / 4

= 47 / 4

= 11.75 milliseconds

6) Counting Semaphore

→ Limits the use of resources by a fixed number of processes / threads

→ It maintains a count between zero and a maximum value.

→ It limits the usage of the resource to the maximum value of the count supported by it.

→ The count associated with a semaphore is decremented by 1 when process / thread acquires.

→ The count is incremented by one when a process / thread releases the semaphore object.

→ The state of the counting semaphore object is set to signal when the count of the object is greater than zero.

Binary Semaphore

→ Implements exclusive access to shared resource by allocating the resource to a single process at a time and not allowing the other processes to access it when it is being used by a process.

→ Only one process / thread can have binary semaphore at a time.

→ The implementation of binary semaphore is OS Kernel dependent.

→ Under OS kernel it is referred as mutex.

→ The state of binary semaphore object is set to signalled when it is not owned by any process.

→ The state of binary semaphore object is set to non-signaled when it is owned by any process / thread.

5) Out of Circuit Programming

→ Out of Circuit Programming is performed outside the target board.

→ The processor or memory chip into which the firmware needs to be embedded is taken out of the target board and it is programmed with the help of programming device.

→ It is a ~~device~~ dedicated unit which contains the necessary hardware circuit to generate the programming signals.

→ The programming device contains a ZIF socket with locking pin to hold the device to be programmed.

→ The programming device will be under the control of a utility program running on a PC.

→ Usually the programmer is interfaced to the PC through RS-232C/USB/Parallel Port Interface.

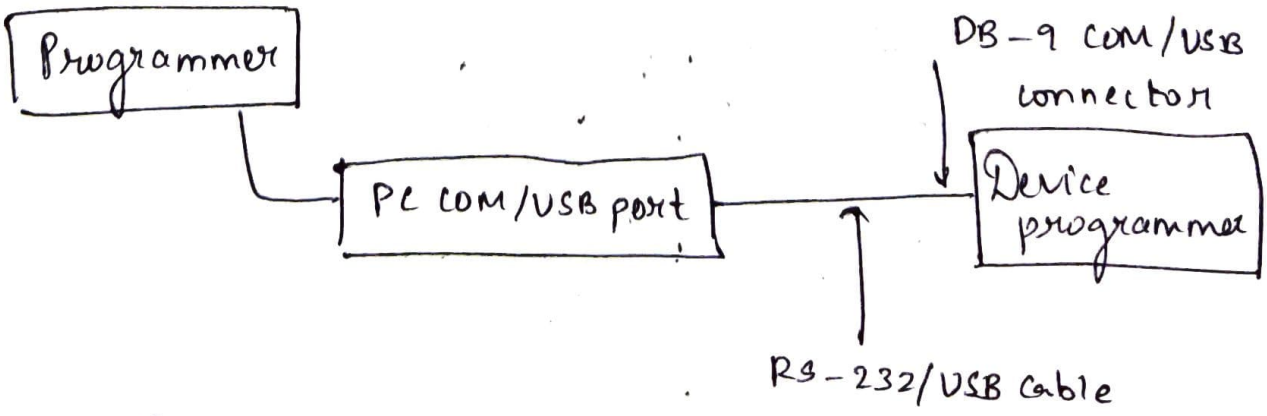


fig. Interfacing of Device Programmer with PC

→ After the firmware is successfully embedded into the device.

→ Insert the device into the board, power up the board and test it for the required functionalities.

→ The major drawback of out-of-circuit programming is the high development time.

→ Out-of-system programming technique is used for firmware integration for low end embedded products which runs without an operating system. It is commonly used for development of low volume products and (Poc) product Development.

* In System Programming (ISP)

→ With ISP, programming is done 'within the system', meaning the firmware is embedded into the target device without removing it from the target board.

→ It is the most flexible and easy way of firmware embedding.

→ Only pre-requisite is that the target device must have an ISP support.

→ Apart from the target board, PC, ISP cable and ISP utility, no other additional hardware is required for ISP.

→ The communication between the target device ~~to communicate with an external hac~~ and ISP utility will be in a serial formal.

→ ISP mode allows the device to communicate with an external host through a serial interface, such as a PC or terminal.

→ The device receives commands and data from the host, erases and reprograms code memory according ~~to~~ to the received command.

→ Once the ISP operations are completed, the device is re-configured so that it will operate normally by applying a reset or a re-power up.