REAL TIME OPERATING SYSTEMS

Lesson-5:

Memory Management Functions

1. Memory allocation

Memory allocation

- When a process is created, the memory manager allocates the memory addresses (blocks) to it by mapping the process-address space.
- Threads of a process share the memory space of the process

Memory Management after Initial Allocation

- Memory manager of the OS─ secure, robust and well protected.
- No memory leaks and stack overflows
- Memory leaks means attempts to write in the memory block not allocated to a process or data structure.
- Stack overflow means that the stack exceeding the allocated memory block(s)

Memory Managing Strategy for a system

- Fixed-blocks allocation
- Dynamic -blocks Allocation
- Dynamic Page-Allocation
- Dynamic Data memory Allocation

Memory Managing Strategy for a system

- Dynamic address-relocation
- Multiprocessor Memory Allocation
- Memory Protection to OS functions

2. Memory allocation in RTOSes

Memory allocation in RTOSes

- RTOS may disable the support to the dynamic block allocation, MMU support to dynamic page allocation and dynamic binding as this increases the latency of servicing the tasks and ISRs.

Memory allocation in RTOSes

- RTOS may not support to memory protection of the OS functions, as this increases the latency of servicing the tasks and ISRs.
- User functions are then can run in kernel space and run like kernel functions

Memory allocation in RTOSes

- RTOS may provide for disabling of the support to memory protection among the tasks as this increases the memory requirement for each task

3. Memory manager functions

Memory Manager functions

- (i) use of memory address space by a process,
- (ii) specific mechanisms to share the memory space and
- (iii) specific mechanisms to restrict sharing of a given memory space

Memory Manager functions

- (iv) optimization of the access periods of a memory by using an hierarchy of memory (caches, primary and external secondary magnetic and optical memories).
- Remember that the access periods are in the following increasing order: caches, primary and external secondary magnetic and then or optical.

4. Fragmentation Memory Allocation Problems

Fragmented not continuous memory addresses in two blocks of a process

- Time is spent in first locating next free memory address before allocating that to the process.
- A standard memory allocation scheme is to scan a linked list of indeterminate length to find a suitable free memory block.
- When one allotted block of memory is de-allocated, the time is spent in first locating next allocated memory block before de-allocating that to the process.

Fragmented not continuous memory addresses in two blocks of a process

- The time for allocation and de-allocation of the memory and blocks are variable (not deterministic) when the block sizes are variable and when the memory is fragmented.
- In RTOS, this leads to unpredicatble task performance

5. Memory management Example

RTOS μCOS-II

- Memory partitioning
- A task must create a memory partition or several memory partitions by using function OSMemCreate ( )
- Then the task is permitted to use the partition or partitions.
- A partition has several memory blocks.

RTOS μCOS-II

- Task consists of several fixed size memory blocks.
- The fixed size memory blocks allocation and de-allocation time takes fixed time (deterministic).
- OSMemGet ( )– to provide a task a memory block or blocks from the partition
- OSMemPut ( ) – to release a memory block or blocks to the partition

Summary

We learnt

- Memory manager allocates memory to the processes and manages it with appropriate protection.
- Static and dynamic allocations of memory.
- Manager optimizes the memory needs and memory utilization

We learnt

- An RTOS may disable the support to the dynamic block allocation, MMU support to dynamic page allocation and dynamic binding as this increases the latency of servicing the tasks and ISRs.
- An RTOS may or may not support memory protection in order to reduce the latency and memory needs of the processes.
- An RTOS may provide of running user functions in kernel space

End of Lesson 5 of Chapter 8