**Spyrtos**

**a Real Time Operating System**

**Abishek KR**

**Abishek R**

**Shubham Saha**

**Spyrtos Documentation**

This documentation explains the design and implementation of Spyrtos.

Spyrtos has a pre-emptive, priority-based scheduler. It is written for microcontrollers using ARM Cortex M4 based processors with hardware and software features to run an OS.

It is written in C and small portion of assembly language code.

The book contains the source code for the operating system with suitable explanation to help run their application in an embedded environment.

**Real Time Systems Concepts**

Operating systems can be classified as Soft and Hard Operating Systems. If the logical correctness is established but there is no stipulated deadline for the process to complete, then it is called Soft OS. If there is a strict deadline restriction to maintain timing correctness, then it is called Hard OS.

Spyrtos is a Soft RTOS.

**Critical Section of Code**

A Critical Section is a code segment that accesses shared variables and must be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.

**Multitasking**

It is the concurrent execution of multiple tasks. Tasks share common processing resources like Central Processing Unit and main memory. Multitasking interrupts the execution of currently running tasks to gain ownership of CPU resource. It is different from parallel programming where a process or multiple process runs parallelly on multiple CPUs. In multitasking more than one process advances in execution over a period.

**Task**

Task or a process is a unit of work. In the sense of multitasking multiple processes are executing at the same time. Task Control Block (TCB) is a data structure used by kernels to store information about each task. Each TCB contains information like Task ID, Priority, Program Counter etc. In addition to that information regarding the process that should be executed next etc are stored. The state of execution is represented by values in registers and flags. Each TCB has memory to store the state variables.

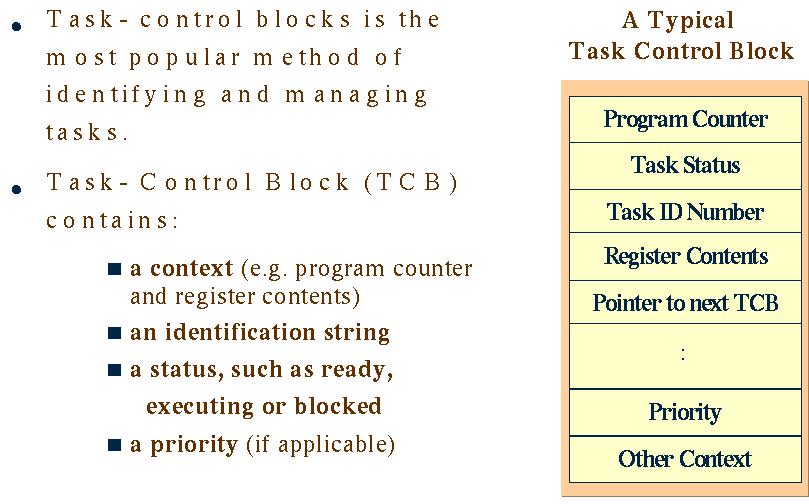


Figure 1: Structure of TCB

Tasks fall into any of the three broader classification.

* Running
* Ready
* Blocked

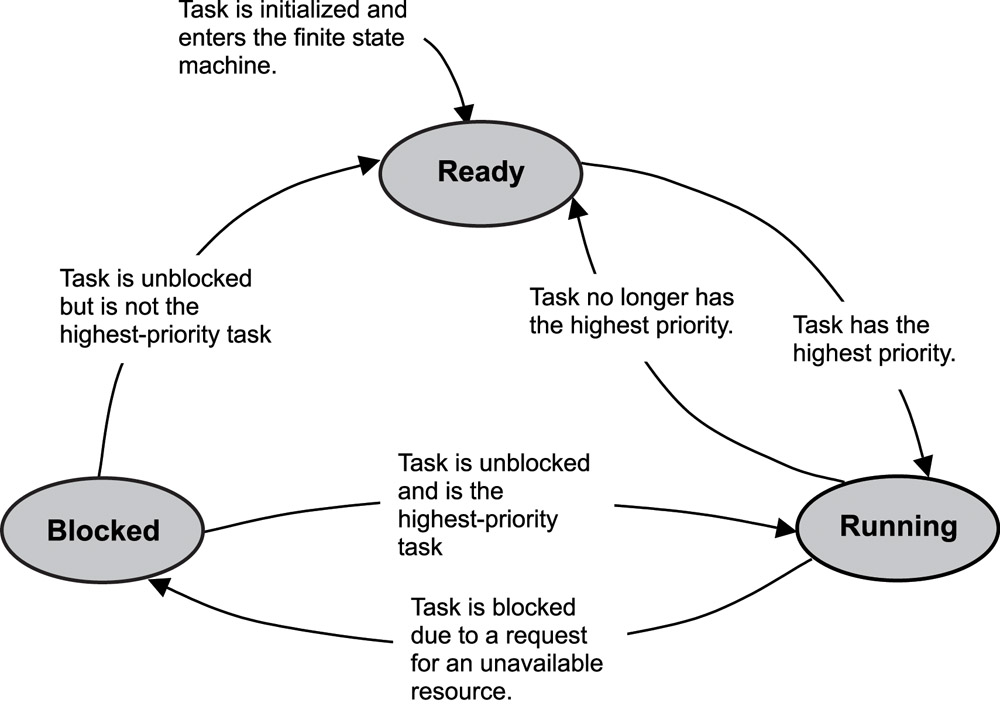


Figure 2: State Diagram depicting typical flow of Processes

READY: The task is being considered for scheduling. The task’s priority is lesser than the currently executing task’s priority.

RUNNING: The task is in control of CPU and being executed.

WAITING: A task which was previously executing is put to hold. It resumes execution after an occurrence of event or freeing up of resource it requires etc.

**Context Switch**

In a multitasking environment a process interrupts the execution of currently running process and gains control of CPU. But the state of execution i.e., Context namely the register values, flags etc must be stored. Only then it can be retrieved at a later point of time to resume execution of the previously interrupted process. The machine context should not be changed. The process of saving and retrieving of context into the designated memory location given by respective TCBs is called the Context Switch. It is advisable to minimize the context switch overhead.

**Source Code**

**Critical Section**

**Entry:**

|  |
| --- |
| start\_critical PROC |
|  |

|  |
| --- |
| EXPORT start\_critical |
|  |

|  |
| --- |
| MRS R0, PRIMASK; |
|  |

|  |
| --- |
| CPSID I; |
|  |

|  |
| --- |
| BX LR |
|  |

ENDP

1. start\_critical PROC: is function start command with the function name “start\_critical”
2. EXPORT start\_critical: The function is available to C functions as well.
3. MRS R0, PRIMASK: Moving contents of PRIMASK, a PSR (Program Status Register) to R0 a GPR (General Purpose Register)
4. CPSID I: The function disables interrupts and all configurable fault handlers
5. BX LR: This returns the function to call location.

Atomicity is achieved by disabling all the interrupts during the execution of critical section.

**Exit:**

|  |
| --- |
| end\_critical PROC |
|  |

|  |
| --- |
| EXPORT end\_critical |
|  |

|  |
| --- |
| MSR PRIMASK, R0 |
|  |

|  |
| --- |
| BX LR |
|  |

|  |
| --- |
|  |
|  |

1. MSR PRIMASK, R0: This instruction is used write values from GPR to PSR. Initial set bit at “I” in PSR will be restored enabling the interrupts.

**Context Switch:**

|  |
| --- |
| os\_first\_task PROC  EXPORT os\_first\_task  CPSID I  LDR R0, =0xE000ED22  LDR R1, =0xFF  STRB R1, [R0]  MOV32 R5, current\_tcb  MOV32 R1, new\_high\_tcb  LDR R2, [R1]  STR R2, [R5]  LDR R0, [R2]  MSR PSP, R0  MRS R0, CONTROL  ORR R0, R0, #2  MSR CONTROL, R0  ISB  LDMFD SP!, {R4-R11}  LDMFD SP!, {R0-R3}  LDMFD SP!, {R12, LR}  LDMFD SP!, {R1, R2}  CPSIE I  BX R1  ENDP |

|  |
| --- |
|  |
|  |

1. CPSID I : Interrupts are disabled
2. Setting PendSV trigger option as set in the Interrupt Control Register (0xE000ED22)
3. The current\_tcb value is updated with next highest priority task’s TCB i.e., new\_high\_tcb
4. R0 which contains the address of Stack Pointer of new TCB is updated as Process Tack Pointer
5. The Stack Pointer to be used if switched from MSP to PSP to facilitate execution of user applications
6. Restore values from TCB to registers.
7. Enable Interrupts.