

# Earliest Deadline First Scheduling Algorithm

Prepared by: Team6



# Table of Contents

- Project description.
- Changes made in “Tasks.c”.
- Tasks created.
- Verifying the system implementation using analytical methods.
- Simulating the tasks using SimSo offline simulator.
- Simulating using the Keil simulator.

# Project Description

The EDF “Earliest Deadline First” is a scheduling algorithm that adopts a dynamic priority-based preemptive scheduling policy, meaning that the priority of a task can change during its execution, and the processing of any task is interrupted by a request for any higher priority task.

All the changes are implemented in the Tasks.c

The implementation adopts rising the priority of the tasks that have the earliest deadline which is calculated using the task period and the current tick.

## Changes made in “Tasks.c”

```

3056 /* _____ */
3057
3058 /* __EDIT__ START INITIALISE TASK LISTS*/
3059
3060 /* _____ */
3061
3062 #if ( configUSE_EDF_SCHEDULER == 1 )
3063 {
3064     vListInitialise( &xReadyTasksListEDF );
3065 }
3066 #endif
3067
3068 vListInitialise( &xDelayedTaskList1 );
3069 vListInitialise( &xDelayedTaskList2 );
3070 vListInitialise( &xPendingReadyList );
3071
3072 #if ( INCLUDE_vTaskDelete == 1 )
3073 {
3074     vListInitialise( &xTasksWaitingTermination );
3075 }
3076 #endif /* INCLUDE_vTaskDelete */
3077
3078 #if ( INCLUDE_vTaskSuspend == 1 )
3079 {
3080     vListInitialise( &xSuspendedTaskList );
3081 }
3082 #endif /* INCLUDE_vTaskSuspend */
3083
3084 /* Start with pxDelayedTaskList using list1 and the pxOverflowDelayedTaskList
using list2. */
3085
3086 pxDelayedTaskList = &xDelayedTaskList1;

```

---

```

main.c | tasks.c | ports.c | FreeRTOSConfig.h | FreeRTOS.h

```

```

276 /* __EDIT__ Start Add the period of the task __EDIT__ */
277
278 /* _____ */
279
280 #if ( configUSE_EDF_SCHEDULER == 1 )
281 {
282     TickType_t xTaskPeriod;
283     #endif
284
285 #if ( ( portSTACK_GROWTH > 0 ) || ( configRECORD_STACK_HIGH_ADDRESS == 1 ) )
286     StackType_t *pEndOfStack; /* Points to the highest valid address for the stack. */
287     #endif
288
289 #if ( portCRITICAL_NESTING_IN_TCB == 1 )
290     UBaseType_t uxCriticalNesting; /* Holds the critical section nesting depth for ports that do not maintain their own count in the port layer. */
291     #endif
292
293 #if ( configUSE_TRACE_FACILITY == 1 )
294     UBaseType_t uxTaskNumber; /* Stores a number that increments each time a TCB is created. It allows debuggers to determine when a task has been created. */
295     UBaseType_t uxTaskCurrentNumber; /* Stores a number specifically for use by third party trace code. */
296     #endif
297
298 #if ( configUSE_MUTEXES == 1 )
299     UBaseType_t uxBasePriority; /* The priority last assigned to the task - used by the priority inheritance mechanism. */
300     UBaseType_t uxCurrentBase;
301     #endif
302
303 #if ( configUSE_APPLICATION_TASK_TAG == 1 )
304     TaskHookFunction_t pxTaskTag;
305     #endif

```

```
0): warning: #188-D: enumerated type mixed with another type
```

```

227 /* Edit Start Adding Tasks To TCB */
228
229 #define prvAddTaskToReadyList( pxTCB )
230 vListInsert( &(amp;ReadyTasksListEDF), &(amp;pxTCB) ->pxStateListItem )
231 #endif
232
233
234 /*
235  * Several functions take an TaskHandle_t parameter that can optionally be NULL,
236  * where NULL is used to indicate that the handle of the currently executing
237  * task should be used in place of the parameter. This macro simply checks to
238  * see if the parameter is NULL and returns a pointer to the appropriate TCB.
239  */
240 #define prvGetTCBFromHandle( pxHandle ) ( ( (pxHandle) == NULL ) ? pxCurrentTCB : (pxHandle) )
241
242 /* The item value of the event list item is normally used to hold the priority
243  of the task to which it belongs (coded to allow it to be held in reverse
244  priority order). However, it is occasionally borrowed for other purposes. It
245  is important its value is not updated due to a task priority change while it is
246  being used for another purpose. The following bit definition is used to inform
247  the scheduler that the value should not be changed - in which case it is the
248  responsibility of whichever module is using the value to ensure it gets set back
249  to its original value when it is released. */
250 #if (configUSE_16_BIT_TICKS == 1)
251 #define taskEVENT_LIST_ITEM_VALUE_IN_USE 0x0000U
252 #else
253 #define taskEVENT_LIST_ITEM_VALUE_IN_USE 0x00000000UL
254 #endif
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

# Changes made in "Tasks.c"

```
main.c tasks.c port.c FreeRTOSConfig.h FreeRTOS.h
528 /* __EDIT__ Initialising The period of Task */
529
530 /* _____ */
531
532 pxNewTCB->xTaskPeriod = period;
533
534 prvInitialiseNewTask( pxTaskCode, pcName, ( uint32_t ) usStackDepth, pvParameters, uxPriority, pxCreatedTask, pxNewTCB, NULL );
535
536 /* __insert the period value in the generic list item before to add the task __ */
537 currentTick = xTaskGetTickCount();
538
539 listSET_LIST_ITEM_VALUE( &(amp; pxNewTCB->xStateListItem), ( pxNewTCB->xTaskPeriod + currentTick );
540
541 prvAddNewTaskToReadyList( pxNewTCB );
542
543 xReturn = pdPASS;
544
545 else
546 {
547     xReturn = errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY;
548 }
549
550 return xReturn;
551 }
552
553 #endif /* configSUPPORT_DYNAMIC_ALLOCATION */
554 /* _____ */
```

```
main.c tasks.c port.c FreeRTOSConfig.h FreeRTOS.h
3227 /* __EDIT__ Check For Tasks that has the Earliest Dead Line */
3228
3229 /* _____ */
3230 #if ( configUSE_EDF_SCHEDULER == 0 )
3231 {
3232     taskSELECT_HIGHEST_PRIORITY_TASK(); /*lint !e9079 void * is used as this macro is used with timers and co-r
3233 }
3234 #else
3235 {
3236     pxCurrentTCB = ( TCB_t * ) listGET_OWNER_OF_HEAD_ENTRY( &( xReadyTasksListEDF ) );
3237 }
3238 #endif
3239 traceTASK_SWITCHED_IN();
3240
3241 /* After the new task is switched in, update the global errno. */
3242 #if ( configUSE_POSIX_ERRNO == 1 )
3243 {
3244     FreeRTOS_errno = pxCurrentTCB->iTaskErrno;
3245 }
3246 #endif
3247
3248 #if ( configUSE_NEWLIB_REENTRANT == 1 )
3249 {
3250     /* Switch Newlib's _impure_ptr variable to point to the _reent
3251     structure specific to this task.
3252     See the third party link http://www.nadler.com/embedded/newlibandFreeRTOS.html
3253     for additional information. */
3254     _impure_ptr = &( pxCurrentTCB->xNewLib_reent );
3255 }
3256 }
```

```
main.c tasks.c port.c FreeRTOSConfig.h FreeRTOS.h
2145 /* __EDIT__ INITIALIZING IDLE TASK */
2146
2147 TickType_t initIdlePeriod = ( TickType_t ) configIDLE_TSM_PERIOD;
2148 xReturn = xTaskPeriodicCreate( prvIdleTask,
2149     configIDLE_TASK_NAME,
2150     configMINIMAL_STACK_SIZE,
2151     ( void * ) NULL,
2152     ( taskIDLE_PRIORITY | portPRIVILEGE_BIT ),
2153     &IdleTaskHandle,
2154     initIdlePeriod );
2155
2156 #else
2157 {
2158     xReturn = xTaskCreate( prvIdleTask,
2159     configIDLE_TASK_NAME,
2160     configMINIMAL_STACK_SIZE,
2161     ( void * ) NULL,
2162     portPRIVILEGE_BIT, /* In effect ( taskIDLE_PRIORITY | portPRIVILEGE_BIT ), but taskIDLE_PRIORITY is zero. */
2163     &IdleTaskHandle ); /*lint !e661 MISRA exception, justified as it is not a redundant explicit cast to all sup
2164 }
2165 #endif
2166
2167 #endif /* configSUPPORT_STATIC_ALLOCATION */
2168
2169 #if ( configUSE_TIMERS == 1 )
2170 {
2171     if( xReturn == pdPASS )
2172     {
2173         xReturn = xTimerCreateTimerTask();
2174     }
2175 }
```

```
main.c tasks.c port.c FreeRTOSConfig.h FreeRTOS.h
3058 /* __EDIT__ START INITIALISE TASK LISTS */
3059
3060 /* _____ */
3061 #if ( configUSE_EDF_SCHEDULER == 1 )
3062 {
3063     vListInitialise( &xReadyTasksListEDF );
3064 }
3065 #endif
3066
3067 vListInitialise( &xDelayedTaskList1 );
3068 vListInitialise( &xDelayedTaskList2 );
3069 vListInitialise( &xPendingReadyList );
3070
3071 #if ( INCLUDE_vTaskDelete == 1 )
3072 {
3073     vListInitialise( &xTasksWaitingTermination );
3074 }
3075 #endif /* INCLUDE_vTaskDelete */
3076
3077 #if ( INCLUDE_vTaskSuspend == 1 )
3078 {
3079     vListInitialise( &xSuspendedTaskList );
3080 }
3081 #endif /* INCLUDE_vTaskSuspend */
3082
3083 /* Start with pxDelayedTaskList using list1 and the pxOverflowDelayedTaskList
3084 using list2. */
3085 pxDelayedTaskList = &xDelayedTaskList1;
3086 pxOverflowDelayedTaskList = &xDelayedTaskList2;
3087 }
```

# Tasks created

```
main.c tasks.c port.c FreeRTOSConfig.h FreeRTOS.h
184 }
185 }
186 void Periodic_Transmitter( void *pvParameters )
187 {
188     const char *eventString = "UART Poll Working Periodically\n";
189     TickType_t xLastWakeTime;
190     xLastWakeTime = xTaskGetTickCount();
191     for(;;)
192     {
193         xQueueSend( EventQueue, &eventString, portMAX_DELAY );
194         vTaskDelayUntil( &xLastWakeTime, 100 );
195     }
196 }
197 void UART_Receiver( void *pvParameters )
198 {
199     const char *eventString;
200     TickType_t xLastWakeTime;
201     xLastWakeTime = xTaskGetTickCount();
202     for(;;)
203     {
204         if ( xQueueReceive( EventQueue, &eventString, NULL ) )
205         {
206             vSerialPutString( ( const signed char* ) eventString, strlen( eventString ) );
207         }
208         vTaskDelayUntil( &xLastWakeTime, 20 );
209     }
210 }
211 void Load_1_Simulation( void *pvParameters )
212 {
213     w32_Counter = NULL;
214 }
215
```

```
main.c tasks.c port.c FreeRTOSConfig.h FreeRTOS.h
151 }
152 void Button_1_Monitor( void *pvParameters )
153 {
154     uint8_t u8_PressFlag = pdFALSE;
155     uint8_t ButtonState;
156     const char *eventRisingString = "\n Button 1 Rising Edge\n";
157     const char *eventFallingString = "\n Button 1 Falling Edge\n";
158     TickType_t xLastWakeTime;
159     xLastWakeTime = xTaskGetTickCount();
160     for(;;)
161     {
162         ButtonState = GPIO_read( PORT_0, PIN0 );
163         if ( ( ButtonState == pdTRUE ) && ( u8_PressFlag == pdFALSE ) )
164         {
165             xQueueSend( EventQueue, &eventRisingString, portMAX_DELAY );
166             u8_PressFlag = pdTRUE;
167         }
168         else if ( ( ButtonState == pdFALSE ) && ( u8_PressFlag == pdTRUE ) )
169         {
170             xQueueSend( EventQueue, &eventFallingString, portMAX_DELAY );
171             u8_PressFlag = pdFALSE;
172         }
173         else
174         {
175             /* Do Nothing */
176         }
177         vTaskDelayUntil( &xLastWakeTime, 50 );
178     }
179 }
180 void Button_2_Monitor( void *pvParameters )
181 {
182 }
183
```

```
main.c tasks.c port.c FreeRTOSConfig.h FreeRTOS.h
152 }
153 void Button_2_Monitor( void *pvParameters )
154 {
155     uint8_t u8_PressFlag = pdFALSE;
156     uint8_t ButtonState;
157     const char *eventRisingString = "\n Button 2 Rising Edge\n";
158     const char *eventFallingString = "\n Button 2 Falling Edge\n";
159     TickType_t xLastWakeTime;
160     xLastWakeTime = xTaskGetTickCount();
161     for(;;)
162     {
163         ButtonState = GPIO_read( PORT_0, PIN1 );
164         if ( ( ButtonState == pdTRUE ) && ( u8_PressFlag == pdFALSE ) )
165         {
166             xQueueSend( EventQueue, &eventRisingString, portMAX_DELAY );
167             u8_PressFlag = pdTRUE;
168         }
169         else if ( ( ButtonState == pdFALSE ) && ( u8_PressFlag == pdTRUE ) )
170         {
171             xQueueSend( EventQueue, &eventFallingString, portMAX_DELAY );
172             u8_PressFlag = pdFALSE;
173         }
174         else
175         {
176             /* Do Nothing */
177         }
178         vTaskDelayUntil( &xLastWakeTime, 50 );
179     }
180 }
181
```

```
main.c tasks.c port.c FreeRTOSConfig.h FreeRTOS.h
216 }
217 void Load_1_Simulation( void *pvParameters )
218 {
219     w32_Counter = NULL;
220     TickType_t xLastWakeTime;
221     xLastWakeTime = xTaskGetTickCount();
222     for(;;)
223     {
224         if ( w32_Counter == NULL, w32_Counter < 37400, w32_Counter++ )
225         {
226             /*Heavy Load Simulation*/
227             vTaskDelayUntil( &xLastWakeTime, 20 );
228         }
229     }
230 }
231 void Load_2_Simulation( void *pvParameters )
232 {
233     w32_Counter = NULL;
234     TickType_t xLastWakeTime;
235     xLastWakeTime = xTaskGetTickCount();
236     for(;;)
237     {
238         if ( w32_Counter == NULL, w32_Counter < 39700, w32_Counter++ )
239         {
240             /*Heavy Load Simulation*/
241             vTaskDelayUntil( &xLastWakeTime, 100 );
242         }
243     }
244 }
245
```

# Verifying the system implementation using analytical methods

1. Calculating the Hyper-Period:

$$\text{Hyper period (H)} = \text{LCM}(P_i) = 100\text{ms}$$

2. Calculating the CPU load:

$$U = R/C = (0.01 * 2 + 0.01 * 2 + 0.01 * 1 + 0.01 * 5 + 5 * 10 + 12) / 100 = 0.621 = 62.1\%$$

3. Check system schedulability using URM:

$$U = 0.01/50 + 0.01/50 + 0.01/100 + 0.01/20 + 5/10 + 12/100 = 0.621$$

$$\text{URM} = 6 * (2^{(1/6)} - 1) = 0.735$$

$$U < \text{URM}$$

So the system is schedulable

# Verifying the system implementation using analytical methods

4. Check system schedulability using Time demand analysis:

Tasks Priority:

Task\_5 > Task\_4 > Task\_1, Task\_2 > Task\_3, Task\_6

Task\_1 schedulability:

$W(50) = 0.01 + 50/20 * 0.01 + 50/10 * 5 = 25.04 < 50$  (Task\_1 is schedulable)

Task\_2 schedulability:

$W(50) = 25.04 < 50$  (Task\_2 is schedulable)



# Verifying the system implementation using analytical methods

Task\_3 schedulability:

$$W(100) = 0.01 + (100/50) * 0.01 + (100/50) * 0.01 + (100/20) * 0.01 + (100/10) * 5 = 50.1 < 100 \quad (\text{Task\_3 is schedulable})$$

Task\_4 schedulability:

$$W(20) = 0.01 + (20/10) * 5 = 10.01 < 20 \quad (\text{Task\_4 is schedulable})$$

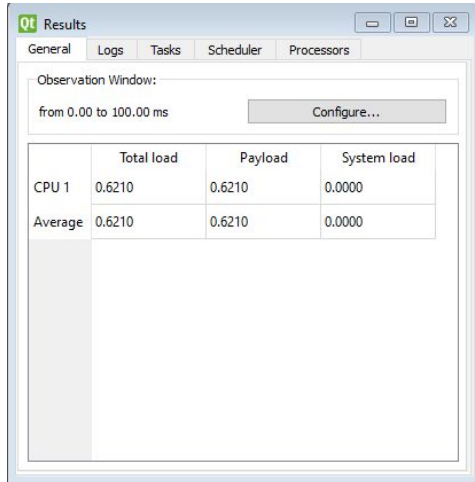
Task\_5 schedulability:

$$W(10) = 5 < 10 \quad (\text{Task\_5 is schedulable})$$

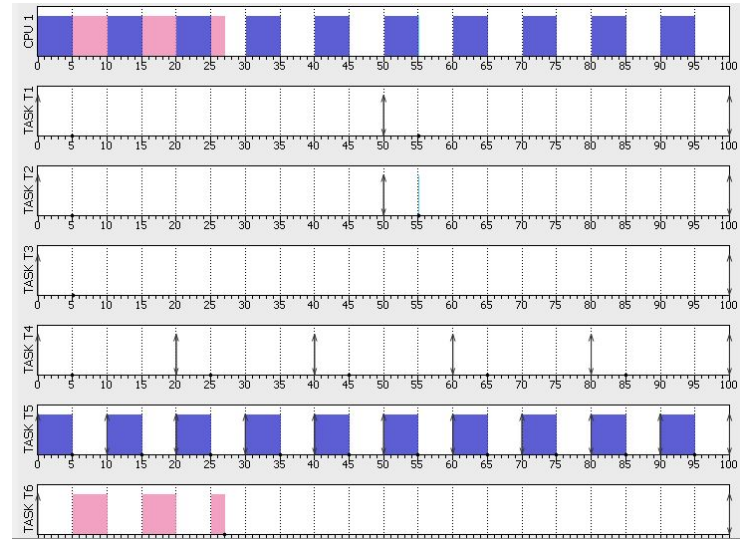
Task\_6 schedulability:

$$W(100) = 12 + (100/5) * 0.01 + (100/5) * 0.01 + (100/20) * 0.01 + (100/10) * 5 = 62.09 < 100 \\ (\text{Task\_6 is schedulable})$$

# Simulating the tasks using SimSo offline simulator



General	Scheduler	Processors	Tasks							
id	Name	Task type	Abort on miss	Act. Date (ms)	Period (ms)	List of Act. dates (ms)	Deadline (ms)	WCET (ms)	Followed by	Priority
1	TASK T1	Periodic	<input checked="" type="checkbox"/> Yes	0.0	50.0	-	50.0	0.01	▼	2
2	TASK T2	Periodic	<input checked="" type="checkbox"/> Yes	0	50	-	50	0.01	▼	2
3	TASK T3	Periodic	<input checked="" type="checkbox"/> Yes	0	100	-	100	0.01	▼	1
4	TASK T4	Periodic	<input checked="" type="checkbox"/> Yes	0	20	-	20	0.01	▼	3
5	TASK T5	Periodic	<input checked="" type="checkbox"/> Yes	0	10	-	10	5	▼	4
6	TASK T6	Periodic	<input checked="" type="checkbox"/> Yes	0	100	-	100	12	▼	1



# Simulating using the Keil simulator

Watch 1		
Name	Value	Type
 Cpu_Load	62.8229027	float
 (PORT0 & 0x00040000) >> 18	0x00000000	ulong
 (PORT0 & 0x00040000) >> 18	0x00000000	ulong
<Enter expression>		
UART #1   UART #2   Watch 1   Watch 2		