EECS 461, Winter 2023, Problem Set 6: SOLUTIONS[1]

1. (a) No. If an interrupt occurs during the computation, then the result may be based on a mix of old and new time information.

   (b) In the suggested example 5:59:59, if the interrupt occurs immediately after the value of hours is read in, then the time will change to 6:00:00. The processor will compute seconds since midnight as though the actual time were 5:00:00, and the result will be 59 minutes and 59 seconds off.

   Alternately, suppose that the time is 23:59:59, the value of seconds is read in first, and an interrupt occurs before the minutes and hours are read in. Then the processor will read both hours and minutes as zero, and it will appear that the number of seconds since midnight is equal to 59, which is incorrect since the number of seconds since midnight is now zero (since the time has changed to midnight while the computation was in progress). Of course the number of seconds since the previous midnight is 86400, so if you think it is still yesterday, the error will be much larger.

   (c) Disable and enable interrupts before and after computing seconds since midnight. There are several ways to do this, for example,

```
long lSecondsSinceMidnight (void)
{
      long lReturnVal;

      disable ( );

         lReturnVal = (((iHours * 60) + iMinutes) * 60 ) + iSeconds;

     enable ( ) ;

     return (lReturnVal);
}
```

   Note that the following way will not work, because the interrupts will never get enabled:

```
long lSecondsSinceMidnight (void)
{
     disable ( );
         return( (((iHours * 60) + iMinutes) * 60 ) + iSeconds);
     enable ( ) ;
}
```

2. No. The problem is that the `vSetTimeZone` function assumes that the `iHours` variable doesn't change while the function is doing its work. If the interrupt routine changes `iHours` during the period of time that interrupts are enabled in the middle of `vSetTimeZone`, that change will get lost when `vSetTimeZone` writes to `iHours` at the end. The local variable `iHoursTemp` does not change the fact that `vSetTimeZone` won't work unless `iHours` doesn't change while the function is calculating.

3. That depends how you define *atomic*. When the main routine changes `fTaskCodeUsingTempsB`, that use can certainly be interrupted. However, the interrupt routine only cares whether the value is zero, and the change from zero to nonzero and back again can't be interrupted. As the code is written, it doesn't even matter whether the use of `fTaskCodeUsingTempsB` is atomic; even if it is not, the value of `fTaskCodeUsingTempsB` cannot get corrupted, and it will still protect the temperature arrays properly.

4. There are two problems with the code. First, both functions must use the same semaphore. Second, both functions must both take and release the semaphore. The changes are shown below:

[1] Revised November 7, 2022.

```
static int iRecordCount;

void increment_records (int iCount)
{
  OSSemGet (SEMAPHORE_COUNT);   /* Use one sem */
  iRecordCount += iCount;
  OSSemGive (SEMAPHORE_COUNT);                /* Add this */
}

void decrement_records (int iCount)
{
  OSSemGet (SEMAPHORE_COUNT);                 /* Add this */
  iRecordCount -= iCount;
  OSSemGive (SEMAPHORE_COUNT); /* Use one sem */
}
```

5. The answer is as shown below. Note that all of the time that `iFixValue` uses `iTemp` as a temporary value to modify `iValue` (which is shared by all of the tasks that call `iFixValue`), it must be protected by a semaphore.

```
static int iValue;

int iFixValue (int iParm)
{
   int iTemp;

   OSSemGet (SEMAPHORE);                       /* Get it here */

   iTemp = iValue;
   iTemp += iParm * 17;
   if (iTemp > 4922)
      iTemp = iParm;
   iValue = iTemp;

   OSSemGive (SEMAPHORE);                      /* Release it here. */

   iParm = iTemp + 179;
   if (iParm < 2000)
      return 1;
   else
      return 0;
}
```

6. Consider the problem of scheduling four tasks, with periods and execution times given as follows:

$$\begin{aligned} T_1: &\quad P_1 = 100, \quad e_1 = 20 \\ T_2: &\quad P_2 = 150, \quad e_2 = 30 \\ T_3: &\quad P_3 = 210, \quad e_3 = 80 \\ T_4: &\quad P_4 = 400, \quad e_4 = 100. \end{aligned}$$

(a) Calculate the total utilization, U, for these four tasks. Do these tasks satisfy the sufficient condition

$$U < n(2^{1/n} - 1)$$

for RMS schedulability?

SOLUTION:

The utilization for each task is given by $U_i = e_i/P_i$, giving

$$U_1 = 0.2,$$
$$U_2 = 0.2,$$
$$U_3 = 0.381,$$
$$U_4 = 0.25.$$

Hence the total utilization is $U = 1.031$. The upper bound is given by $4(2^{1/4} - 1) = 0.76$. Since the upper bound is violated, the sufficient condition is not satisfied, and the test is inconclusive.

(b) If your answer to the preceding question is negative, does the answer change if you consider only the first three tasks? By eliminating tasks do you ever arrive at a combination that does satisfy the sufficient condition for schedulability?

SOLUTION: If we consider only the first three tasks, then the total utilization is $U_1+U_2+U_3 = 0.781$. The upper bound is given by $3(2^{1/3}-1) = 0.7798$, and thus the sufficient condition is still not satisfied.

If we consider only the first two tasks, then the total utilization is $U_1 + U_2 = 0.4$. The upper bound is given by $2(2^{1/2} - 1) = 0.8284$, and thus the sufficient condition for RMS schedulability is satisfied.

(c) For each task, sketch the function

$$W_i(t) = \sum_{k=1}^{i-1} \left\lceil \frac{t}{P_k} \right\rceil e_k + e_i$$

that determines the amount of time the CPU spends executing the tasks $T_1, \ldots, T_i$ in the interval $[0, t]$. You may wish to modify the m-file "PS6_prob6a.m".

SOLUTION: Using the Matlab m-file "PS6_prob6a_soln.m" yields the plots depicted in Figure 1.

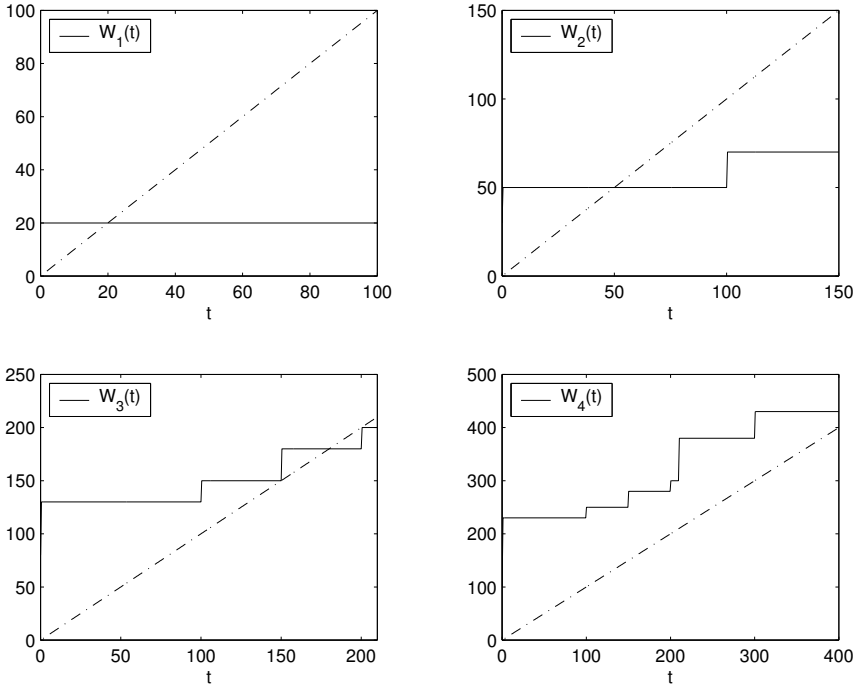

Figure 1: Plots of $W_i(t)$ to Determine RMS Schedulability

(d) Determine which tasks satisfy the necessary and sufficient condition for RMS schedulability. Recall that this condition requires the existence of a time $t^* \leq P_i$ for which $W_i(t^*) \leq t^*$.

SOLUTION:

It follows from Figure 1 that tasks T_1, T_2, and T_3 are all schedulable, but that T_4 is not.

(e) For those tasks that are schedulable, determine the times at which they complete.

SOLUTION:

It follows from Figure 1 that task T_1 completes at $t = 20$, task T_2 completes at $t = 50$, and task T_3 completes (barely) at $t = 150$.

(f) The task scheduler will switch back and forth between the tasks as they run. Plot which task is running at which times. (Modify the m-file "PS6_prob6e.m".)

SOLUTION:

Using the Matlab m-file "PS6_prob2e_soln.m" yields the plots in Figure 2.

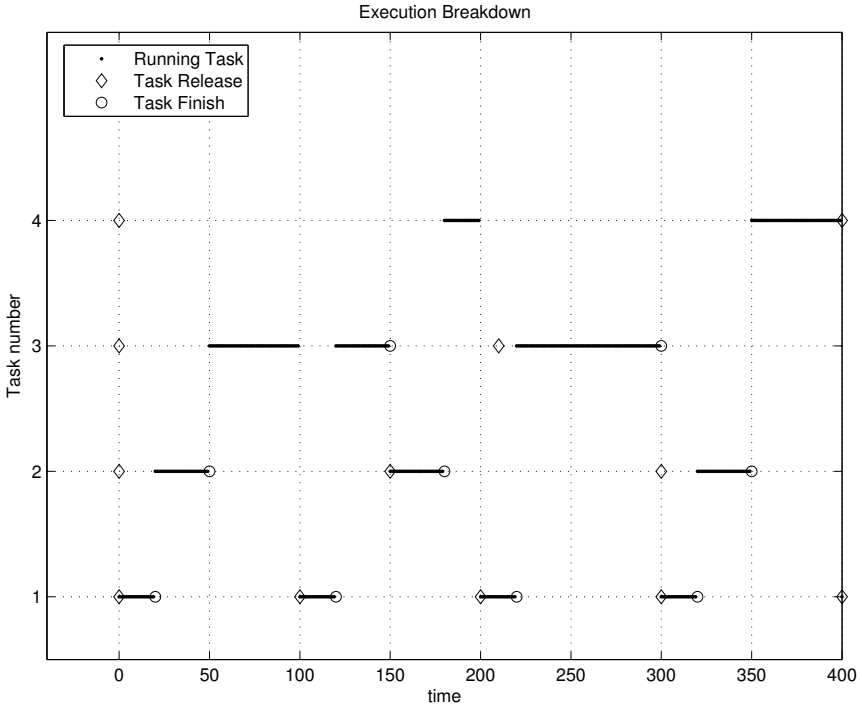

Figure 2: Plot showing when each task is running.

7. Consider the problem of scheduling three tasks, each with zero phasing and with periods and execution times shown in Table 1. The deadline for each task is equal to its period.

Task	Period	Execution Time
1	$P_1 = 200$	$e_1 = 50$
2	$P_2 = 400$	$e_2 = 100$
3	$P_3 = 600$	$e_3 = 300$

Table 1: Three Periodic Tasks

(a) Compute the total utilization for these three tasks. Do they meet the sufficient condition for Rate Monotonic (RM) schedulability? Explain.

ANSWER: The utilizations for the individual tasks are

$$U_1 = e_1/P_1 = 0.25, \qquad U_2 = e_2/P_2 = 0.25, \qquad U_3 = e_3/P_3 = 0.5, \tag{1}$$

and thus the total utilization satisfies

$$U = 1. \tag{2}$$

The sufficient condition for RM schedulability with three tasks requires that

$$U \leq 3(2^{1/3} - 1) \approx 0.7798, \tag{3}$$

and is thus violated. As a result, the utilization test provides no information about RM schedulability.

(b) Are Tasks 1 and 2 RM schedulable? Explain your answer using a plot of $W_2(t)$ vs. t. You may sketch this plot in Figure 3.

ANSWER: Tasks 1 and 2 are RM schedulable. The plot of $W_2(t)$ in Figure 3 shows that Task 2 completes at $t = 150$ seconds, before the 400 second deadline.

(c) Are Tasks 1, 2, and 3 RM schedulable? Explain your answer using a plot of $W_3(t)$ vs. t. You may sketch this plot in Figure 4.

ANSWER: Tasks 1,2, and 3 are not all schedulable. The plot of $W_3(t)$ in Figure 4 shows that Task 3 has not completed before its deadline of $t = 600$ seconds.

(d) In Figure 5, sketch the times at which the three tasks will be running under the rate monotonic scheduling protocol.

ANSWER: The plots are shown in Figure 5. Note that Task 3 does not complete by its deadline.

(e) An alternative to the rate monotonic scheduling protocol is the earliest deadline first (EDF) scheduling protocol. Using EDF, the task with the closest deadline is given the highest priority, and will preempt any other task. In Figure 6, sketch the times at which the three tasks will be running under the EDF protocol. Do all three tasks complete by their first deadlines?

ANSWER: The running times of the three tasks under the EDF protocol are sketched in Figure 6. Note that each task meets its first deadline. An alternate solution, also correct, is shown in Figure 7. The reason for the two solutions is that at $t = 400$ seconds, both Tasks 1 and 3 need to run, and they are equally far away from their deadline. How this ambiguity is resolved would depend on the specific RTOS implementation.

To determine whether the tasks are EDF schedulable, we would need to either invoke the utilization test $U \leq 1$, which is satisfied in this case, or we need to plot the execution times out to the least common multiple of the task periods, when all the tasks would once again need to start executing at the same time.

SOLUTION TO PROBLEM 7:

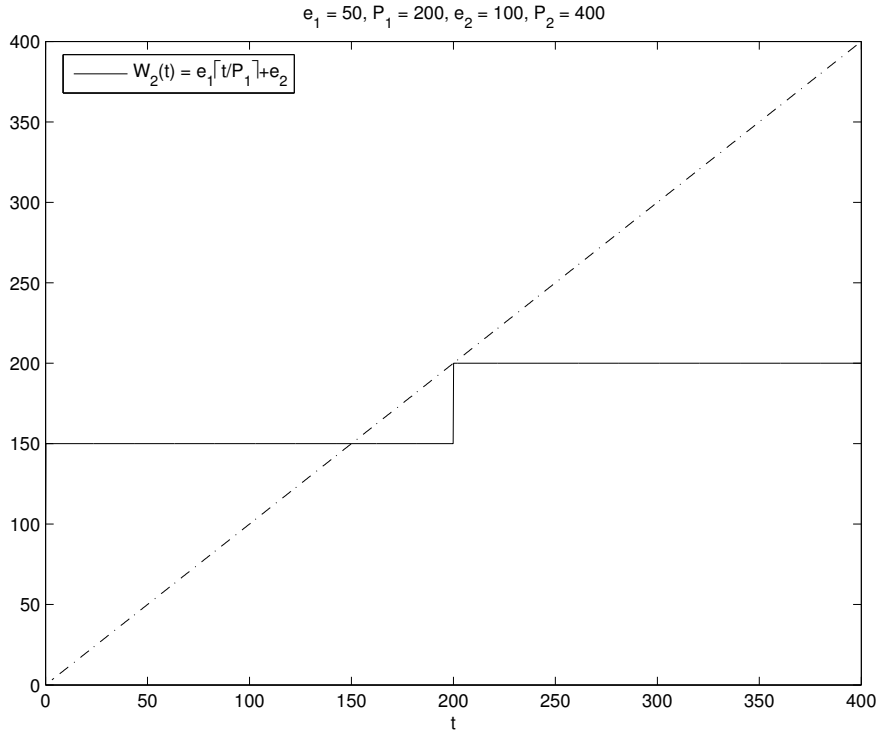

Figure 3: Schedulability for Tasks 1 and 2

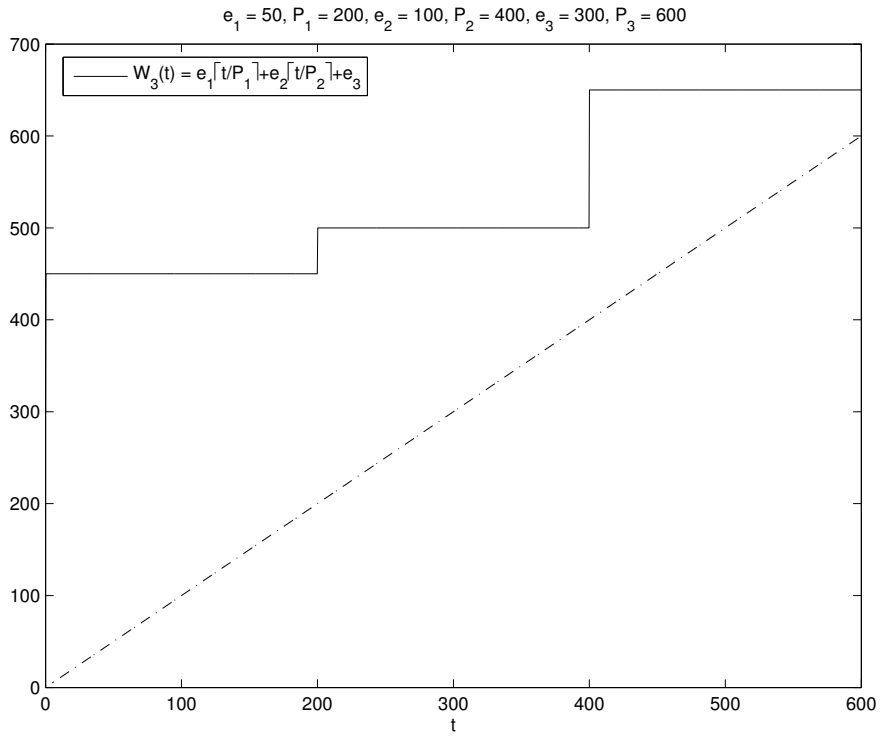

Figure 4: Schedulability for Tasks 1,2, and 3

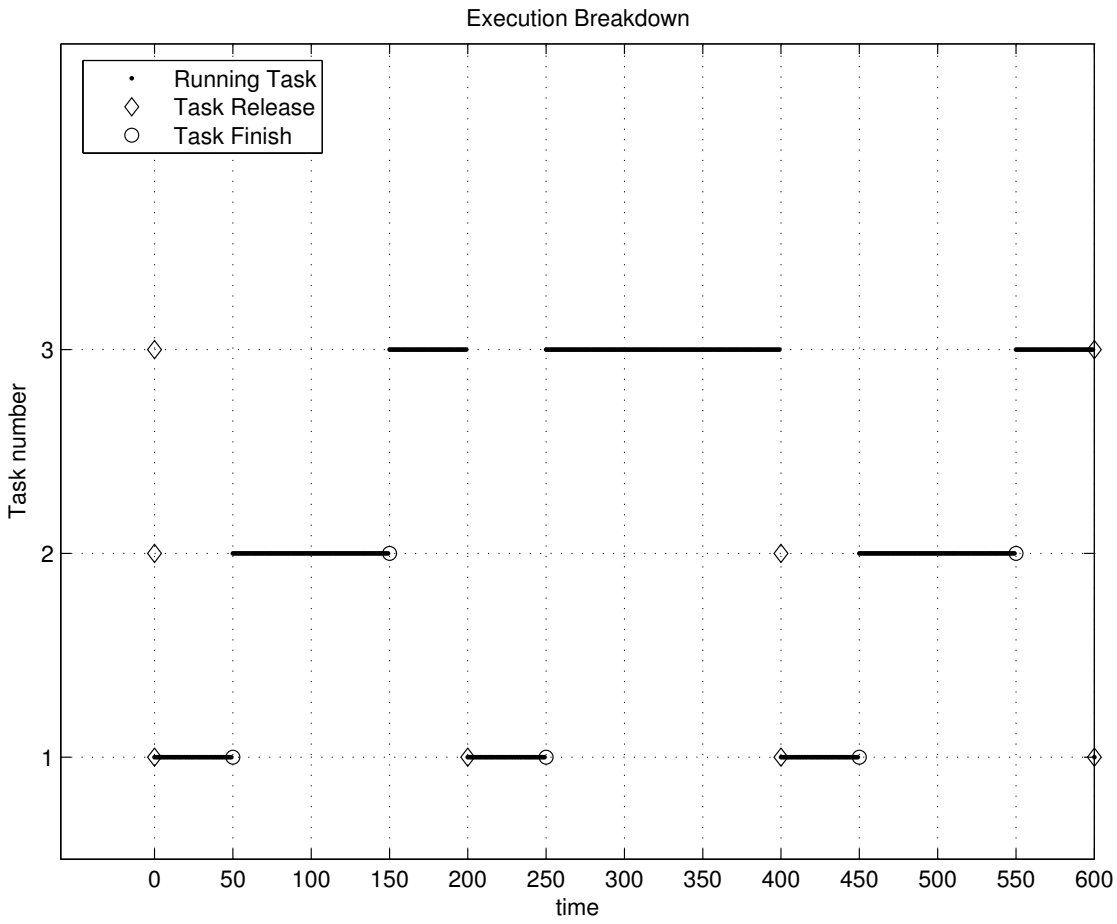

Figure 5: Run Times for Tasks 1, 2, and 3 under Rate Monotonic Scheduling

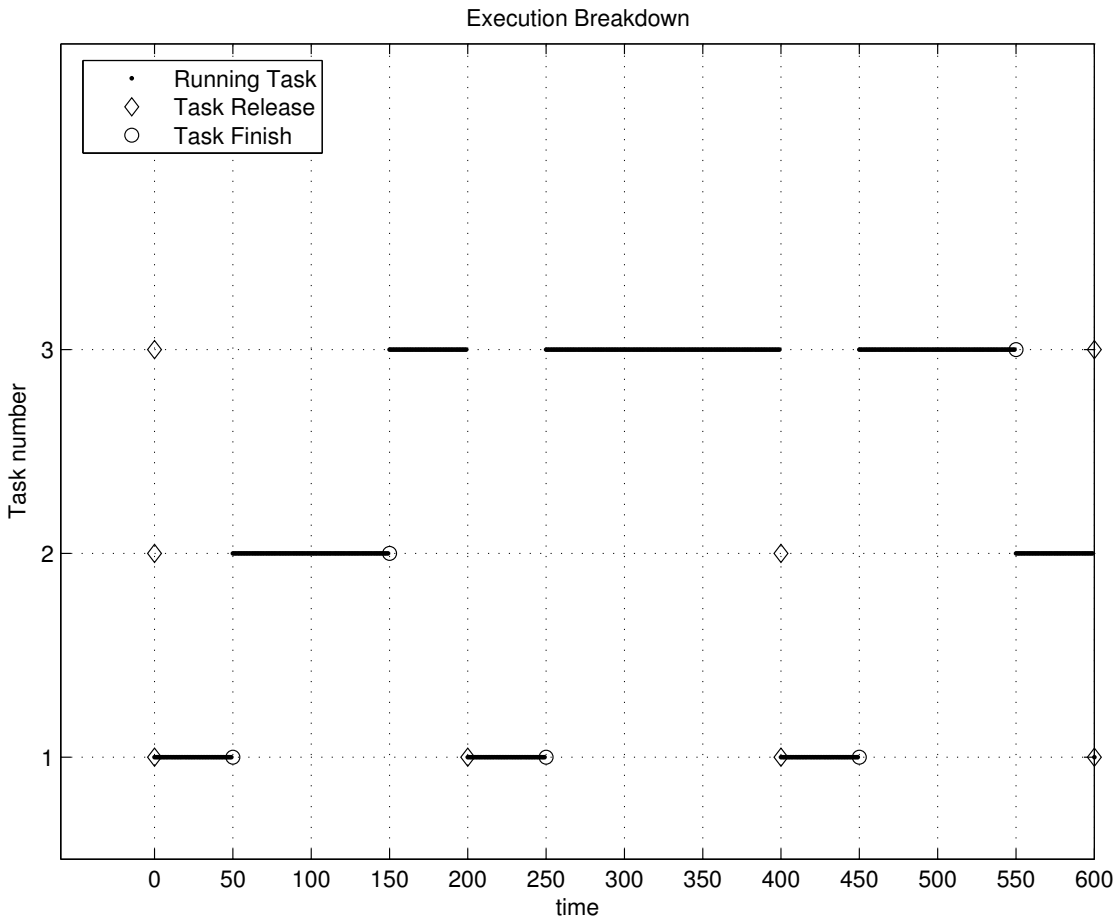

Figure 6: Run Times for Tasks 1, 2, and 3 under Earliest Deadline First Scheduling, Solution 1

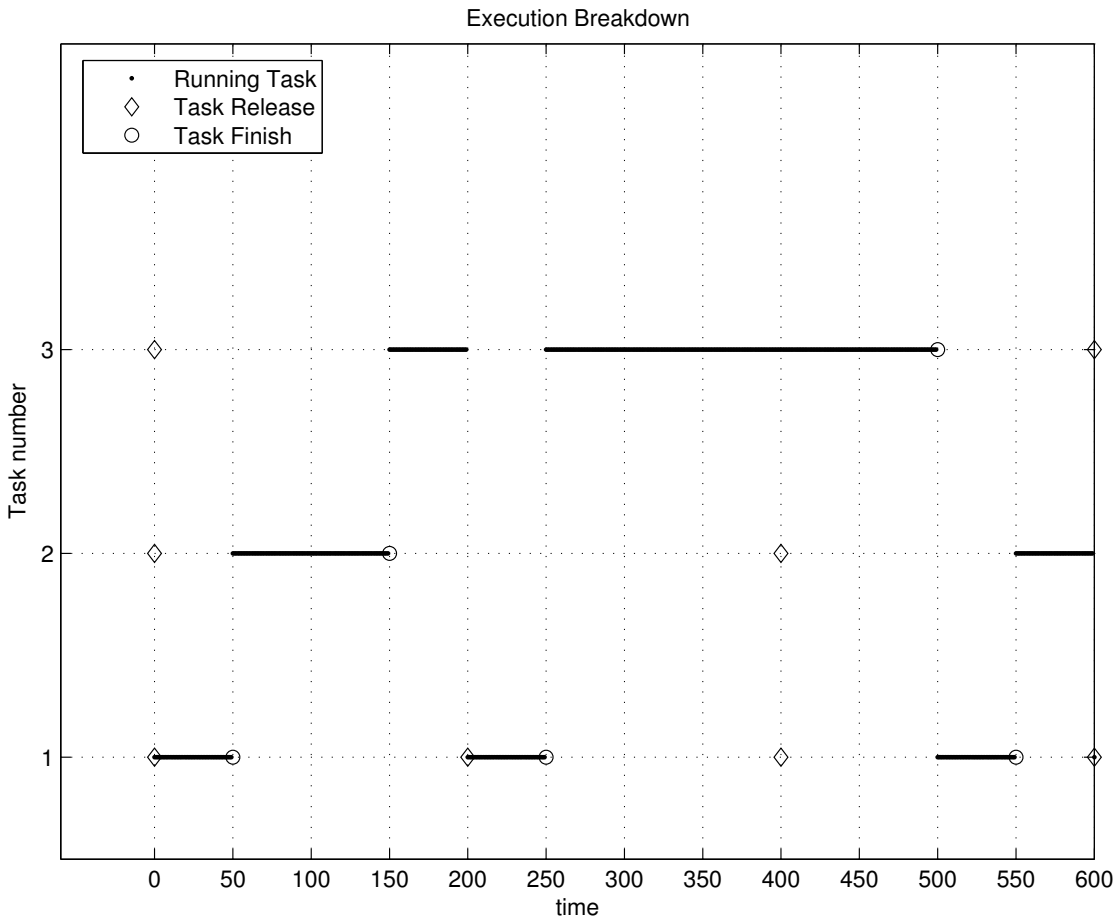

Figure 7: Run Times for Tasks 1, 2, and 3 under Earliest Deadline First Scheduling, Solution 2