Proceedings of the 1987 Winter Simulation Conference
A. Thesen, H. Grant, W. David Kelton (eds.)

ALTERNATIVES FOR MODELING OF PREEMPTIVE SCHEDULING

James O. Henriksen
Wolverine Software Corporation
7630 Little River Turnpike
Suite 208
Annandale, VA 22003-2653

ABSTRACT

A system which includes overt instances of preemptive scheduling, or which requires the use of preemptive scheduling to model the system, can pose difficulties to a modeler. By preemptive scheduling we mean having to reschedule or cancel a previously scheduled event (or activity completion). The difficulties in modeling preemptive scheduling stem from the highly stylized constructs which simulation languages provide for such operations. This paper describes these difficulties, reviews an approach which can reduce the difficulties for some applications, and presents an approach which can be used to model preemptive scheduling, using only simple simulation language constructs.

1. ORGANIZATION OF THIS PAPER

Section 2 gives examples of systems which require the use of preemptive scheduling. Modeling difficulties common to SLAM II (Pritsker, 1986), SIMAN (Pegden, 1986), and GPSS/H (Henriksen & Crain, 1983) are described for these systems. Section 3 reviews an approach which reduces the extent to which preemptive scheduling is used for modeling certain types of conveyor systems. This approach was first described in (Henriksen & Schriber, 1986) and in (Henriksen, 1986). Section 4 presents an alternative approach which eliminates preemptive scheduling altogether. While not universally applicable, the approach is an interesting tool for the modeler to add to his or her toolkit.

2. DIFFICULTIES IMPOSED BY PREEMPTIVE SCHEDULING

2.1 Systems Which Require Preemptive Scheduling

Preemptive scheduling is frequently required for modeling systems which contain prioritized activities. For example, in a hospital, providing medical attention to patients is (or ought to be) a higher priority activity than administrative activities, such as filling out forms. Furthermore, attending to a patient who has had a heart attack is of higher priority than attending to a patient who has broken a finger. When modeling a hospital, the modeler must deal with situations in which a previously scheduled event, e.g., completion of the treatment of a broken finger, must be rescheduled because one or more servers (doctors, nurses, etc.) are interrupted to provide service to a patient of higher priority.

The hospital exemplifies the most commonplace form of preemptive scheduling, in which a previously scheduled event (or completion of an activity) must be delayed. Typically, the rescheduling takes into account the remaining time to complete the activity, plus some time for the overhead of interrupting affected servers. In other types of systems, the scheduled occurrence of an event may have to be moved up in time or eliminated altogether. For example, in a model of a factory, a breakage event might have been scheduled for a specific tool on a machine. In the absence of any other events, the breakage would occur as scheduled. If, however, the tool were changed prior to the breakage, the breakage event would have to be canceled. To carry this example a little further, assume that the tool breakage depended on the rate of operation of the machine. If the machine were to be speeded up, the breakage would be likely to occur earlier. Therefore, a scheduled tool breakage would have to be moved up in time.

In the examples presented thus far, preemptive scheduling has been an overt property of the system to be modeled. In some modeling applications, preemptive scheduling may be used as a modeling convenience, rather than to represent an overt characteristic of the system. For example, consider a factory which operates for two shifts per day. For the modeler, it may be convenient to schedule certain events without regard to time-of-day, and to schedule an 8-hour interruption of all servers at the conclusion of the second shift. (The ignoring of startup and shutdown overheads implied by this simplified approach would probably introduce unacceptable modeling inaccuracies for all but the most primitive models.)

2.2 Language Constructs for Preemptive Scheduling

In SLAM II and SIMAN network (block diagram) models, and in all GPSS/H models, preemptive scheduling is accomplished by means of PREEMPT blocks. (Unless otherwise noted, when we refer to SLAM II or SIMAN, we restrict ourselves to their network (block diagram) modeling capabilities.) The same block name, or node type, is used in all

three languages. (In GPSS/H, preemptive scheduling can also be accomplished by means of the FUNAVAIL block. For our present purposes, the FUNAVAIL block can be ignored.)

In all three languages, the operation of the PREEMPT block is directly tied to a single server; i.e., scheduled events can be canceled only for units of traffic (called entities in SLAM II and SIMAN and transactions in GPSS/H) which currently possess a single server (a resource of capacity 1 in SLAM II and SIMAN, a facility in GPSS/H). In a SLAM II or SIMAN model, a PREEMPTed entity is rerouted to a specified network node (block). In GPSS/H, the PREEMPT block provides many options for handling PREEMPTed transactions. For example a PREEMPTed transaction can be rerouted to a specified block, as in SLAM II and SIMAN. Alternatively, a transaction's use of a facility can be suspended until the PREEMPTing transaction completes its use of the facility (indicated by its execution of a RETURN block). Other options exist for totally removing the transaction from contention for the facility, storing the time remaining for its scheduled use of the facility, etc.

In the world view of SLAM II and SIMAN, entities are very dynamic objects (they crawl around the newtork (block diagram)), and resources are static objects. Similarly, in GPSS/H, transactions are dynamic objects, and facilities are static. This gives rise to the difficulty in inmplementing preemptive scheduling in these languages: to be interrupted, an inherently dynamic object must be directly associated with a static object. An example of this difficulty is given in the next section.

3. REDUCING THE EXTENT TO WHICH PREEMPTIVE SCHEDULING IS USED

Suppose that one wanted to model the flow of objects on a nonaccumulating conveyor. A nonaccumulating conveyor is a conveyor for which the distance between objects on the conveyor is fixed; i.e., objects cannot "pile up" behind an object whose motion is blocked (the belt cannot slide under blocked objects). When a nonaccumulating conveyor stops, by definition, all objects on the conveyor stop. Further suppose, for the sake of argument, that for SLAM II and SIMAN, the models are to be developed using basic network language constructs, i.e., not taking advantage of language extensions which provide conveyor entities.

In the most straightforward modeling approach that could be taken with SLAM II, SIMAN and GPSS/H, objects to be moved on a conveyor would be represented as entities (in SLAM II and SIMAN) or transactions (in GPSS/H). When an object is placed on the conveyor, its scheduled time of arrival at its destination (assuming no intervening blockages) could be calculated and the object could undergo an explicit time delay. The mechanisms by which explicitly scheduled time delays are modeled are language-specific: in SLAM II, an activity-on-branch approach is used; in SIMAN, the DELAY block is used; and in GPSS/H, the ADVANCE block is used. (All three languages also include capabilities for modeling indefinite time delays, where completion of the delay period is indicated by occurrence of a particular event. Conceivably, these capabilities could be used for the modeling the system under discussion, but for our purposes, we will consider only the use of explicitly scheduled time delays.)

If any object on the conveyor is blocked, all objects on the belt must be stopped. Given our choice of modeling object motion by means of explicit time delay, the only way an object can be stopped is to use a PREEMPT block (node). This in turn implies that each object to be stopped must possess a capacity 1 resource (facility) and that data structures must be maintained to represent (1) the object-resource associations and (2) which objects (or associated resources) are currently on the conveyor. Since objects come and go in dynamic fashion, while resources are fixed in number for a given run, this implies the needs to (1) dynamically associate and dissociate resources to and from objects and (2) to predetermine the maximum number of objects that will be on the conveyor at any given time, in order to fix the size of the pool of resources. This, my friends, is a major programming burden.

Henriksen and Schriber (Henriksen & Schriber, 1986) described a technique they called the "follow-the-leader" approach for modeling nonaccumulating conveyors. Under their approach, an explicit time delay is calculated only for the "leader" object on a conveyor. The "leader" object is the object furthest downstream on the conveyor. Any objects which are upstream from the leader object are modeled by using an auxiliary data structure (They used a GPSS/H user chain.) which keeps track of (1) which objects are on the conveyor and (2) the time (or spatial) separation between objects. They considered only single entry, single exit conveyor systems, i.e., systems in which all objects enter a conveyor at one end and exit at the other end. Henriksen (Henriksen, 1986) expanded this approach to cover the case where objects could be placed on a nonaccumulating conveyor at randomly selected points.

The beauty of the follow-the-leader approach is that only one resource (facility) is required per nonaccumulating conveyor. Since conveyors do not come and go, modeling them as resources (facilities) is very straightforward. Each time an object enters the nonaccumulating conveyor, a check must be made to determine whether it is the new leader or whether there is another object downstream from it on the conveyor. Similarly, when an object is removed from the conveyor, a check must be made to see whether another object is behind it on the conveyor. If so, a time delay must be scheduled to model the follower object's transit to the exit point. On those occasions when the

conveyor must be stopped, the simultaneous delay of all objects on the conveyor can be accomplished by PREEMPTing the leader object, since the travel times of all other objects are expressed relative to the leader.

To summarize, what this approach does is to reduce the problem of simultaneously suspending the time delays for N objects to the problem of suspending the time delay for a single object, at the cost of some extra (possibly non-trivial) bookkeeping. (Pretty neat, eh?)

4. ELIMINATING PREEMPTIVE SCHEDULING

4.1 The Optimistic Scheduling Approach

In this section we present an alternative modeling approach which eliminates the need for preemptive scheduling, per se. The technique is called the "optimistic scheduling" approach. In the discussion which follows, a moving-object, passive-resource world view is assumed, although the approach described could readily be adapted to other world views. The technique works as follows:

1. Pertinent data which represents or affects timing data is stored in two forms: each object has its own local copy, and all objects share a single global copy. The form this data takes depends on the particular application. An example will be given below.

2. At each point in a model where an explicit time delay which is subject to potential rescheduling (preemption) is scheduled, the pertinent global timing data is updated (if necessary) and copied into the data structure representing the object. For example, in GPSS/H, data values would be copied from savevalues (shared, global) into transaction parameters (local to a transaction).

3. The object for which the delay is scheduled optimistically assumes that its delay will occur as scheduled, i.e., that no interruptions will occur. In the case of a GPSS/H transaction, an ADVANCE block would be executed, naively assuming that the corresponding time delay would take place as scheduled.

4. If subsequent conditions require rescheduling of the delay, the pertinent global data is updated to reflect changed conditions.

5. When the object for which the delay was optimistically scheduled completes the delay, it compares its copy of the pertinent timing data to the current global timing data. If the local and global copies differ, this indicates that the completion time for the delay estimated in step 3 is no longer valid. If they are identical, the object for which the delay was optimistically scheduled can proceed on its merry way.

6. In the event the local and global copies differ, corrective action must be taken. The corrective action may take several forms. In the simplest case, the object for which the delay was optimistically scheduled may be able to react to the changed data on its own behalf. For example, if the scheduled delay has been lengthened since it was originally scheduled, scheduling an additional delay to make up the difference between the optimistic delay and the actual delay may work nicely. Appendix B contains a GPSS/H program which utilizes this approach. This example is discussed in Section 4.2, below.

In some circumstances, the object for which the delay was optimistically scheduled may be incapable of reacting to changed conditions. For example, if the length of an optimistically scheduled delay has been shortened, by the time the delay has completed, it is too late for the object to react. In such cases, the model must take appropriate action at the point at which conditions change. In GPSS/H, the transaction which causes the optimistic delay to be changed might be able to SPLIT off a copy of itself and the offspring transaction could assume the role that was formerly played by the transaction for which the optimistic delay was scheduled. If this approach is used, upon discovering that conditions have changed, the transaction for which the optimistic delay was scheduled can simply disappear from the model (via a TERMINATE block), "knowing" that it has been superseded by another transaction.

4.2 A Sample Model

Appendix A contains a simple example of preemptive scheduling implemented in GPSS/H. The system modeled has a single server which processes two streams of jobs, a low priority job stream and a high priority job stream. If the server is processing a low priority job when a high priority job arrives, processing of the low priority job is suspended; i.e., high priority jobs preempt the processing of low priority jobs. Within a given job stream, jobs are processed in FIFO order. Interarrival times for low priority jobs are uniformly distributed over the interval (50,150), i.e., 100 ± 50. Interarrival times for high priority jobs are uniformly distributed over the interval (0,1200), i.e., 600 ± 600. Service times for both types of jobs are uniformly distributed over the interval (40,120), i.e., 80 ± 40. No overhead time is associated with preemption; i.e., the server is assumed to be capable of instantaneously changing jobs.

Jobs flowing into the model are represented by transactions in two disjoint model segments. Low priority jobs are modeled in the first segment, which is a simple one-line, single-server queueing model, where the server is a facility named

SERVER. Low priority transactions SEIZE the SERVER, hold it (subject to PREEMPTion) for a period of time, and RELEASE it.

High priority jobs are modeled in the second model segment. When a transaction representing a high priority job arrives, it executes a PREEMPT block. If the SERVER facility is neither busy nor already PREEMPTed, the transaction immediately PREEMPTs the facility. If the facility is already PREEMPTed, the transaction must wait until preceding transactions have RETURNed the facility. If the facility is currently SEIZEd (by a low priority job), the low priority transaction is suspended until the high priority job RETURNs the facility. Specifically, at the point of PREEMPTion, GPSS/H records the amount of remaining ADVANCE time for the PREEMPTed job, and when the RETURN block is executed by a high priority job, the time delay for the PREEMPTed job is resumed at its ADVANCE block.

Appendix B contains a GPSS/H model of the same system implemented without the use of preemptive scheduling. One of the unusual features of this model is that it uses a pair of facilities to model one server. SERVER is used to model the processing of low priority jobs, and SERVER2 is used to model the processing of high priority jobs. The use of facility entities guarantees FIFO processing within each job stream. Intervals of time during which both SERVER and SERVER2 are busy (SEIZEd) correspond to periods of PREEMPTion in the Appendix A model.

In order to support the optimistic scheduling approach, the Appendix B model keeps track of the total amount of time spent handling high priority jobs, i.e., the total amount of time that SERVER2 is busy. This total time is accumulated in a savevalue named BUMPTIME. When a transaction has just SEIZEd SERVER, it waits at a GATE block until SERVER2 is not busy. When SERVER2 is no longer busy, the low priority transaction makes a copy (in a transaction parameter named MYCOPY) of the total time that SERVER2 has been busy up to this point in time. The low priority transaction then executes an ADVANCE block, assuming that its randomly sampled service time will hold, i.e., that no high priority jobs will enter the system prior to completion of the service interval of the low priority transaction.

When the low priority transaction emerges from its ADVANCE block, it compares the value stored in its MYCOPY parameter to the current value of BUMPTIME. If the BUMPTIME value exceeds, the MYCOPY value, the single server has been preempted in the system, but not PREEMPTed in the model. Accordingly, additional time delay must take place to model the interruption. The amount of additional delay required is precisely the difference between the BUMPTIME and MYCOPY values. Prior to executing an ADVANCE block with the appropriate delay, the low priority transaction updates its local copy (MYCOPY) of BUMPTIME.

Since additional delay could occur while the transaction is making up the difference between the BUMPTIME and MYCOPY values, testing for preemption and incurring additional delay when appropriate is implemented in the form of a loop. A low priority transaction completes its service when its last optimistically scheduled delay interval expires without intervening interruption.

4.3 When to Use Optimistic Scheduling

The optimistic scheduling approach has limitations. It works inefficiently if interruptions of service occur too frequently. Furthermore, for simple forms of preemptive scheduling, using built-in simulation language capabilities may be easier than implementing the optimistic scheduling approach. For example, the Appendix A model is actually easier to write and read than the Appendix B model, for the GPSS/H user who understands how PREEMPT and RETURN work. However, in more complex models, the use of built-in language capabilities may be unacceptably complex. Just understanding how the software works may require a great deal of reading. On the other hand, the optimistic scheduling approach is built upon lower level language capabilities, readily understood by non-expert users.

5. CONCLUSIONS

Section 2 gave examples of systems which require the use of preemptive scheduling and presented some of the modeling difficulties imposed by such systems. Section 3 showed that with a little bit of thought, systems which would appear to require wholesale use of preemptive scheduling can sometimes be viewed from a perspective which reduces the amount of preemptive scheduling. Finally, Section 4 presented the optimistic scheduling approach, which allows modeling preemptive scheduling without the use of complex built-in language constructs for preemption.

6. ACKNOWLEDGEMENTS

Thanks go to Bob Crain and Jim Harbour for their thoughtful suggestions and assistance in the preparation of this paper.

APPENDIX A: A GPSS/H MODEL WHICH USES PREEMPTIVE SCHEDULING

```
GPSS/H VAX/VMS RELEASE 0.98 (UG217)     25 AUG 1987   16:39:31     FILE: USEPREE.GPS

LINE# BLOCK#  *LOC    OPERATION       A,B,C,D,E,F,G   COMMENTS

    1                 SIMULATE
    2          *
    3          *      MODEL TO ILLUSTRATE PREEMPTIVE SCHEDULING
    4          *
    5          PTIMORD TABLE          M1,40,10,20            MEASURE TIME THRU THE SYSTEM
    6          PTIMEHI TABLE          M1,40,10,20            DITTO FOR HIGH PRIORITY
    7
    8          ORDSERV FUNCTION       RN2,C2                 ORDINARY SERVICE
    9          0,40/1,120
   10          HISERV  FUNCTION       RN3,C2                 HIGH PRIORITY SERVICE
   11          0,40/1,120
   12          *
   13          *      "ORDINARY" JOB SEGMENT
   14          *
   15       1         GENERATE        100,50                 ORDINARY JOBS ARRIVE
   16       2         QUEUE           ORDQ                   "ORDINARY" QUEUE
   17       3         SEIZE           SERVER                 GRAB THE SERVER
   18       4         DEPART          ORDQ                   EXIT "ORDINARY" QUEUE
   19       5         ADVANCE         FN$ORDSERV             "ORDINARY" PROCESSING TIME
   20       6         RELEASE         SERVER                 JOB COMPLETED
   21       7         TABULATE        PTIMORD                RECORD PROCESSING TIME
   22       8         TERMINATE       0                      EXIT THE SYSTEM
   23          *
   24          *      HIGH PRIORITY JOB SEGMENT
   25          *
   26       9         GENERATE        600,600
   27      10         QUEUE           HIGHQ                  HIGH PRIORITY QUEUE
   28      11         PREEMPT         SERVER                 BUMP LOWER PRIORITY JOB (IF ANY)
   29      12         DEPART          HIGHQ                  EXIT HIGH PRIORITY QUEUE
   30      13         ADVANCE         FN$HISERV              HIGH PRIORITY PROCESSING TIME
   31      14         RETURN          SERVER                 JOB COMPLETED
   32      15         TABULATE        PTIMEHI                RECORD PROCESSING TIME
   33      16         TERMINATE       0                      EXIT THE SYSTEM
   34          *
   35          *      TIMER SEGMENT
   36          *
   37      17         GENERATE        ,,10000                TIMER TRANSACTION
   38      18         TERMINATE       1                      SHUT DOWN THE MODEL
   39                 START           1
   40                 END
```

APPENDIX B: A GPSS/H PROGRAM WHICH ILLUSTRATES OPTIMISTIC SCHEDULING

```
GPSS/H VAX/VMS RELEASE 0.98 (UG217)     25 AUG 1987   16:39:18     FILE: OPTIMIST.GPS

LINE# BLOCK#   *LOC     OPERATION       A,B,C,D,E,F,G   COMMENTS

    1                   SIMULATE
    2           *
    3           *       MODEL TO ILLUSTRATE "OPTIMISTIC" SCHEDULING
    4           *
    5           PTIMORD TABLE           M1,40,10,20             MEASURE TIME THRU THE SYSTEM
    6           PTIMEHI TABLE           M1,40,10,20             DITTO FOR HIGH PRIORITY
    7
    8           ORDSERV FUNCTION        RN2,C2                  ORDINARY SERVICE
    9           0,40/1,120
   10           HISERV  FUNCTION        RN3,C2                  HIGH PRIORITY SERVICE
   11           0,40/1,120
   12           *
   13           *       "ORDINARY" JOB SEGMENT
   14           *
   15       1           GENERATE        100,50,,,,3PL           ORDINARY JOBS ARRIVE
   16       2           QUEUE           ORDQ                    "ORDINARY" QUEUE
   17       3           GATE NU         SERVER2                 WAIT UNTIL NOT "PREEMPTED"
   18       4           SEIZE           SERVER                  GRAB THE SERVER
   19       5           DEPART          ORDQ                    EXIT "ORDINARY" QUEUE
   20       6           ASSIGN          MYCOPY,XL$BUMPTIME,PL   TOTAL PREEMPTED TIME
   21       7           ADVANCE         FN$ORDSERV              "ORDINARY" PROCESSING TIME
   22           *
   23           *       TEST WHETHER ADDITIONAL "PREEMPTED" TIME HAS BEEN ACCUMULATED.
   24           *       IF SO, MAKE UP THE DIFFERENCE.
   25           *
   26       8   MAKEUP  TEST G          XL$BUMPTIME,PL$MYCOPY,EXIT      NO ADD'L TIME => EXIT
   27       9           SAVEVALUE       DIFF,XL$BUMPTIME-PL$MYCOPY,XL   DIFFERENCE TO MAKE UP
   28      10           ASSIGN          MYCOPY,XL$BUMPTIME,PL           UPDATE LOCAL INFO
   29      11           ADVANCE         XL$DIFF                         MAKE UP THE DIFFERENCE
   30      12           TRANSFER        ,MAKEUP                         TRY AGAIN
   31
   32      13   EXIT    RELEASE         SERVER                  JOB COMPLETED
   33      14           TABULATE        PTIMORD                 RECORD PROCESSING TIME
   34      15           TERMINATE       0                       EXIT THE SYSTEM
   35           *
   36           *       HIGH PRIORITY JOB SEGMENT
   37           *
   38      16           GENERATE        600,600,,,,3PL          HIGH PRIORITY JOBS ARRIVE
   39      17           QUEUE           HIGHQ                   HIGH PRIORITY QUEUE
   40      18           SEIZE           SERVER2                 FICTITIOUS SECOND SERVER
   41      19           DEPART          HIGHQ                   EXIT HIGH PRIORITY QUEUE
   42      20           ASSIGN          HITIME,FN$HISERV,PL     SERVICE TIME
   43      21           SAVEVALUE       BUMPTIME+,PL$HITIME,XL  ACCUM TOTAL TIME "PREEMPTED"
   44      22           ADVANCE         PL$HITIME               HIGH PRIORITY PROCESSING TIME
   45      23           RELEASE         SERVER2                 JOB COMPLETED
   46      24           TABULATE        PTIMEHI                 RECORD PROCESSING TIME
   47      25           TERMINATE       0                       EXIT THE SYSTEM
   48           *
   49           *       TIMER SEGMENT
   50           *
   51      26           GENERATE        ,,10000                 TIMER TRANSACTION
   52      27           TERMINATE       1                       SHUT DOWN THE MODEL
   53                   START           1
   54                   END
```

REFERENCES

Pegden, C. D. (1986). Introduction to SIMAN with Version 3.0 Enhancements, Systems Modeling Corporation, State College, Pennsylvania.

Pritsker, A. A. B. (1986). Inroduction to Simulation and SLAM II, Third Edition. Halstead Press, John Wiley & Sons, New York.

Henriksen, J. O. (1986). You Can't Beat the Clock: Studies in Problem Solving In: Proceedings of the 1986 Winter Simulation Conference (J. Wilson, S. Roberts, J. Henriksen, eds.). Society for Computer Simulation, San Diego, California, 713-726.

Henriksen, J. O. and Crain, R. C. (1983). GPSS/H User's Manual, Second Edition. Wolverine Software Corporation, Annandale, Virginia.

Henriksen, J. O. and Schriber, T. J. (1986). Simplified Approaches to Modeling Accumulating and Nonaccumulating Conveyor Systems. In: Proceedings of the 1986 Winter Simulation Conference (J. Wilson, S. Roberts, J. Henriksen, eds.). Society for Computer Simulation, San Diego, California, 575-593.

AUTHOR'S BIOGRAPHY

JAMES O. HENRIKSEN is the president of Mr. Henriksen is the president of Wolverine Software Corporation, located in Annandale, Virginia (a suburb of Washington, D.C.) Wolverine Software was founded in 1976 to develop and market GPSS/H, a state-of-the-art version of the GPSS language. Since its introduction in 1977, GPSS/H has gained wide acceptance in both industry and academia. From 1980-1985, Mr. Henriksen served as an Adjunct Professor in the Computer Science Department of the Virginia Polytechnic Institute and State University, where he taught courses in simulation and compiler construction at the university's Northern Virginia Graduate Center. Mr. Henriksen is a member of ACM, SIGSIM, SCS, the IEEE Computer Society, ORSA, and SME.

Mr. Henriksen is a frequent contributor to the literature on simulation. He has given invited presentations at the Winter Simulation Conference, the Summer Simulation Conference, and at the Annual Simulation Symposium. He served as the Business Chairman of the 1981 Winter Simulation Conference and as the General Chairman of the 1986 Winter Simulation Conference.

James O. Henriksen
Wolverine Software Corporation
7630 Little River Turnpike - Suite 208
Annandale, VA 22003-2653
(703) 750-3910