THE BOSS SIMULATOR – AN INTRODUCTION

Paul F. Roth
Manager, Modeling and Simulation
DSSSG Marketing Support
Burroughs Corporation
Paoli, Pennsylvania 19301

Introduction and Summary

This paper introduces BOSS, the Burroughs Operational Systems Simulator. BOSS is a block-diagram-oriented, data-base-driven simulator program, in the general class of GPSS, as opposed to the programming language simulator, as typified by SIMULA. Simulators of the BOSS type are easily utilized by the analyst, with the cost of this facility being the relative lack of generality of expression provided by a language simulator.

Various features make BOSS particularly appealing and useful:

1. Most important, BOSS is very easy to use because it may be employed without formal language programming and the resultant tedium of language debugging and compilation.

2. Modeling is simplified because certain biases and default characteristics in logical flow and queue servicing are inherent in the BOSS view of system operation.

3. The parameters of a model may be mapped on a BOSS block diagram and then transferred directly to input data files.

4. BOSS contains a large variety of data base error messages and notes which facilitate debugging of the model.

5. Output report generation is completely spontaneous and is managed by the BOSS program itself without input commands.

In practice, the modeler maps BOSS parameters onto a logical flow chart which facilitates the coding of a file-oriented data base input directly from this flow chart. The series of input files comprise data input to be executed with BOSS machine code which is obtained by compiling the BOSS program (currently implemented in ALGOL).

BOSS views the world as users (processes) comprised of tasks which contend for resources (units). A process is a model of a logical sequence of tasks, to which must be allocated units and time. Tasks therefore represent the changes of state incurred by the users.

This generalized concept applies to a vast range of system modeling applications.

The simulation of a supermarket system, for example, could model the actions of customers as the "processes" with the "tasks" taken as operations such as shopping and checkout which contend for "units" such as counters, clerks, and carts. BOSS can manage the concurrent handling of many simultaneous shoppers in diverse states of operation.

The simulation of traffic handling at an airport might assign as the processes the incoming and outgoing flights which contend for resources such as terminal space, ground handling equipment, and personnel. Another transportation example could be the simulation of equipment being allocated on a rail system, where the processes are train operations that are scheduled and allocated such resources as motive power and crew.

The simulation of an operating EDP system can be modeled by taking each application program using the system and dividing it into a sequence of program segments (i.e., processing, I/O, and communication operations) each of which can be considered a task requiring the assignment of the various system hardware resources for representative periods of time. Thus the task called "disk access" requires the allocation of a disk control unit, a disk unit, and an I/O channel for a period of time representative of the actual operation of the particular system being modeled. Multiprogramming can be simulated by allowing concurrent processes (the mix) to contend for system resources.

A salient feature of BOSS is the report generator which is programmed to print out statistical data spontaneously at the end of a run and at various time slices during a run at the discretion of the modeler. Representative outputs are:

1. Process completion statistics ("throughput")

2. Resource queuing and utilization statistics

3. Chronology of events.

Thus, if the simulation examples given are referred to, the data would indicate, for example: customer shopping time for a supermarket; utilization of railroad motive power, traffic congestion at an airport terminal, and job throughput of the computer application for given job mixes.

Actual, repeated, field experience shows that a creative modeler can usefully apply BOSS after one or two days of individual instruction in coding and review of case studies. Thus BOSS is a quickly learned medium for the system analyst to communicate his model to the computer for simulation, without requiring the use of a programmer or source language compilation and debugging. To further enhance this capability, BOSS has an ample repertoire of messages and diagnostics for debugging the input data files.

In summary, BOSS is a tool for system simulation which requires no programming, manages its own data reporting, and has a general purpose capability which is especially adaptable for system simulation.

Although BOSS has been developed as a fully general-purpose tool, it has been primarily employed by the Burroughs Corporation as a marketing aid to assess the relative performance of proposed ADP system configurations. Its impact is particularly felt in the simulation of systems responding to real-time loads or incorporating parallel processing techniques such as multiprogramming and multiprocessing. This paper will discuss, in order, the BOSS programmatic features, logical and structural features, and output generation. Finally, the coding for a minimal server model will be demonstrated. The emphasis given in this treatment is on the philosophy and compactness of BOSS.

BOSS-World View

In order to achieve a block diagram simulator with compactness of notation and coding, certain assumptions of how things work are incorporated into the program. BOSS treats a "process-oriented" world, where a process represents the sequence of operations or state changes incurred by the user or customer of a system as he traverses the system in time. The basic operational entity of a process is a task, a discrete operation which consumes time and/or resources. The resources are treated as pools of units, each pool actually being a set of identical interchangeable resource elements. Processes primarily carry the attributes of logical structure, relative priority, and arrival characteristics. Tasks primarily carry the attributes of unit and time specification. Unit sets carry the attributes of multiplicity, dependency, and queue-servicing discipline. When a task starts, it requests units, which may be assigned in two ways: normal mode, where the unit is released back to its pool at task completion time, and transfer mode, where the unit is released to some other pool at completing time. Unit dependency refers to the ability to disable named "dependent" units while a particular "independent" unit is busy. Queuing is incurred by tasks, unable to get a complement of units, going into queue on the (empty) pool. Tasks in queue are serviced by a discipline attributed to the pool. The attribute struction is summarized in Figure 1.

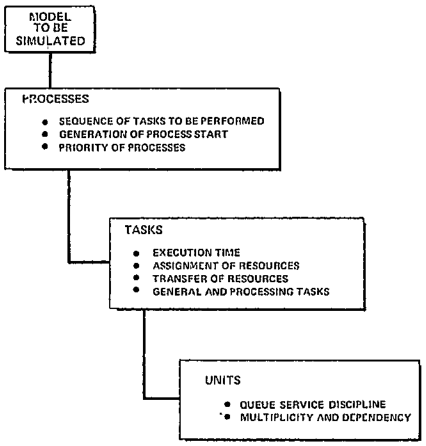

MODEL TO BE SIMULATED
PROCESSES
• SEQUENCE OF TASKS TO BE PERFORMED
• GENERATION OF PROCESS START
• PRIORITY OF PROCESSES
TASKS
• EXECUTION TIME
• ASSIGNMENT OF RESOURCES
• TRANSFER OF RESOURCES
• GENERAL AND PROCESSING TASKS
UNITS
• QUEUE SERVICE DISCIPLINE
• MULTIPLICITY AND DEPENDENCY

Figure 1. BOSS Elements and Attributes

Input and Coding Structure

The BOSS data base is input to the computer as a series of files having the following features:

1. Fixed format, right-hand justified fields.
2. Entirely integer with the exception of several negative-sign and flag characters. There are no verbs or statement capability.
3. Comments are allowed throughout the data base.

There are three primary types of data employed in encoding the data base:

1. Type Identification (TID), numbers, which relate to the process attributes.
2. Selection codes, which invoke logical procedures, etc.
3. Parameters.

These conventions enable a compact modeling notation and facilitate the mapping of information on a block diagram. One advantage of this approach is that changes and improvements made to the program may be made available to the user by increasing the menu of selection codes, without changing the structure of the data base. This permits upward compatibility of BOSS application data bases with an improving program. Another advantage is that the report generator can spontaneously output information for every appropriate TID without resorting to identifying titles or names which would require additional programming.

Type Identification (TID) Number

Attributes of a process are encoded primarily in terms of TID numbers. The numbers themselves provide links between various data files. Every process, uniquely distinguishable in attributes (structure, priority) is given a TID number which is a global identifier.

A process type identifies a particular sequence of tasks. Each task in the process is identified by a sequence number, local to the process. Each task is also given a Task TID number, a global identifier which serves as a link to a file giving the task attributes.

The task attributes are primarily time and units to be allocated. Time is a local quantity to the task and is specified by a selection code (discussed later) and two parameters; units are specified by global TID numbers denoting units pools and linking to a file giving unit attributes.

Units are declared in a file under unit TID numbers. Attributes of units are multiplicity, a parameter giving the initial quantity of units in a pool, and group, a selection code denoting queue servicing discipline or other optional characteristics. Each unit within a pool is given a global unique identification number, which is used to accrue utilization statistics for output reporting.

Selection Codes

Selection codes are used for several purposes: logical branching and merging in the task sequence; algorithm selection to determine parametric values from distrbuted ensembles; and selection of unit queuing attributes, such as queue service discipline.

Logical merging involves the selection of a combinatorial rule for starting a task with one or more predecessor branches.

There are two options for merging:

Type 0	Logical AND
Type 1	Logical OR

Logical branching involves the selection of a procedure which determines the conditions for continuation of flow on completion of a task. There are currently eight options, called "successor codes" which provide a great variety of choice in task logical sequencing, unit allocation, and inter-task communication. Because these represent a very powerful capability of BOSS, they are discussed in a subsequent section of their own.

Finally, selection codes are employed to denote various attributes of unit pools. The present repertoire of code is:

0 - Queue discipline: FIFO, with priority servicing

1 - Queue discipline: non-queuing (multiserver)

2 - Queue discipline: FIFO, regardless of priority

3 - Type 0 queue discipline, with units dedicated to the process to which assigned, for life of process, whether specifically busy or not.

Parameters

Parametric values may be generated by invoking an algorithm which selects a parametric value based on some rule. When a selection code is used to specify the selection algorithm, one or two parameters relating to the algorithm are also given. Table 1 demonstrates codes currently available.

TABLE 1. Selection Codes Currently Available

Selection Code	Parameter 1	Parameter 2	Generated Quantity
1	Value		Constant
2	Minimum Value	Maximum Value	Uniformly Distributed Quantity
3	Mean Value	One Standard Deviation	Pseudo Normally Distributed Quantity
4	Mean Value	Minimum or Maximum Cut-off Value	Exponentially Distributed Quantity

Task Successor Codes

The sequencing of system events flow is controlled and specified by a successor selection code. There are currently eight different types, numbered 0 through 7; these are explained in the following paragraphs and flow charts.

Type 0 - Direct Successor

The completion of the present task directly initiates one or more successor tasks. Up to 10 "parallel" successor tasks may be specified as indicated in Figure 2. All parallel successor tasks are initiated at the same simulated time. If no successor task is assigned, the process ends when the present task is finished.

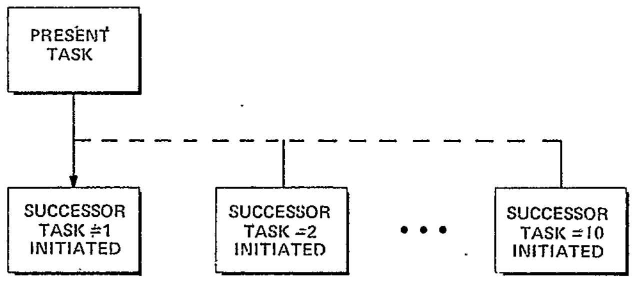

PRESENT TASK
SUCCESSOR TASK #1 INITIATED
SUCCESSOR TASK #2 INITIATED
• • •
SUCCESSOR TASK #10 INITIATED

Figure 2. Direct Successor Tasks

Type 1 - Probabilistic Successor

At the completion of the present task, one successor task is chosen from a group of at least two candidates on the basis of probability, P, expressed in permillage points. Thus, a probability of 0.861 is expressed as 861. The maximum number of candidate successor tasks that can be used with this type of selection is six, as indicated in Figure 3.

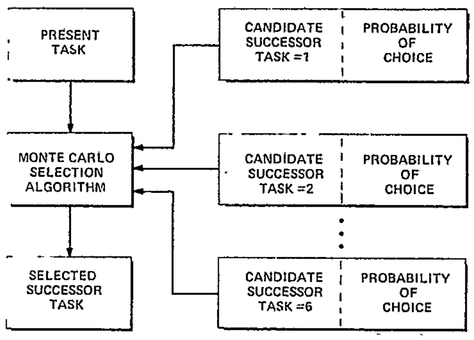

PRESENT TASK
MONTE CARLO SELECTION ALGORITHM
SELECTED SUCCESSOR TASK
CANDIDATE SUCCESSOR TASK #1
PROBABILITY OF CHOICE
CANDIDATE SUCCESSOR TASK #2
PROBABILITY OF CHOICE
CANDIDATE SUCCESSOR TASK #6
PROBABILITY OF CHOICE

Figure 3. Probabalistic Branching

Type 2 - Conditional/Counter

This selection type has two options: A and B.

Option A of Type 2. The completion of the present task gives rise to a test: has the present task been performed the required number of times (N) for this process? If the counter (C) does not equal 0, the test fails, and the flow initiates a successor task (usually reentrant to or upstream of the present task) and the counter is decremented by 1; if the test succeeds (C = 0), the flow initiates an "exit" successor task, as indicated in Figure 4.
The value of N is determined by specifying one of the four available distribution type selection codes.

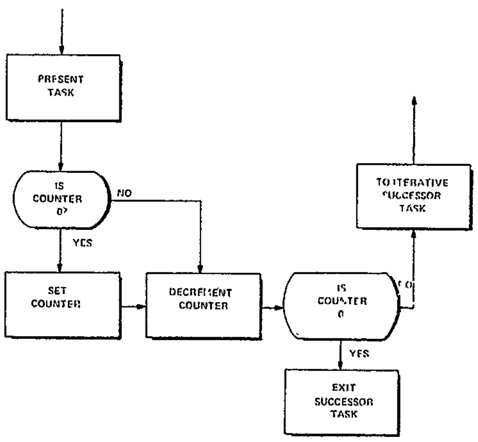


Figure 4. Conditional Successor – Counter Test

Option B of Type 2. At the completion of the present task, one of two successor tasks is chosen, based on comparing the number (N) of individual units existing within a given unit type with a specified number (S), as indicated in Figure 5.

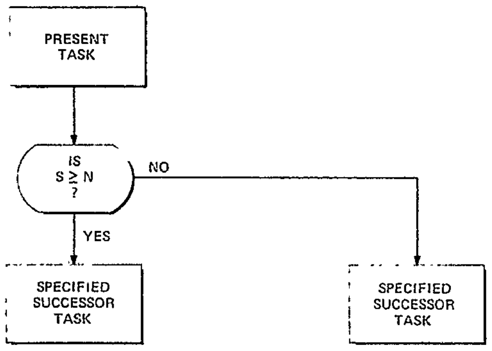


Figure 5. Conditional Successor Unit Test

Type 3 - Enabler

The completion of the present task acts as a pulsed switch to enable, or connect, the flow resulting from the completion of other tasks in branches of the same process as indicated in Figure 6. In this case the present task has an "indirect" successor task. Should the direct predecessor task(s) not be completed prior to completion of the present task, the indirect successor task will not be initiated unless the present task is performed again and is completed after all the direct predecessor task(s) are completed.

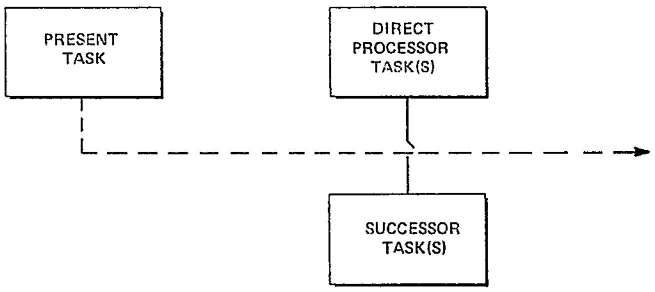


Figure 6. Enabled Successor

Type 4 - Conditional/Queue Test

The completion of the present task causes the following test: will the time (TQ) that the direct successor task must wait in queue for the unit complement exceed a pre-specified value (N), including 0? If the test (the task gets its units without waiting or with a wait less than the selected period) fails, the normal flow continues, as indicated in Figure 7. If the test passes, the process jumps out of the unit queue, bypasses the direct successor task, and flows to the conditional alternate successor task.

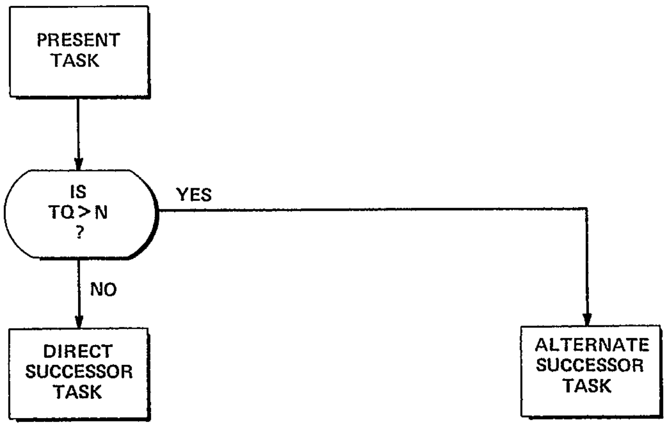


Figure 7. Queue Bypass Successor

Type 5 - Seize Task

Units assigned to the present task are busied as usual. However, these units are not released when the present task is finished, but are held (seized) until a specified release task in the same process is finished at some future time, (see Figure 8). Up to 12 tasks may be specified on the coding form, with the first task entry specifying the release task, and the remaining entries specifying parallel successor codes as with successor code 0. At least one successor task must be specified.

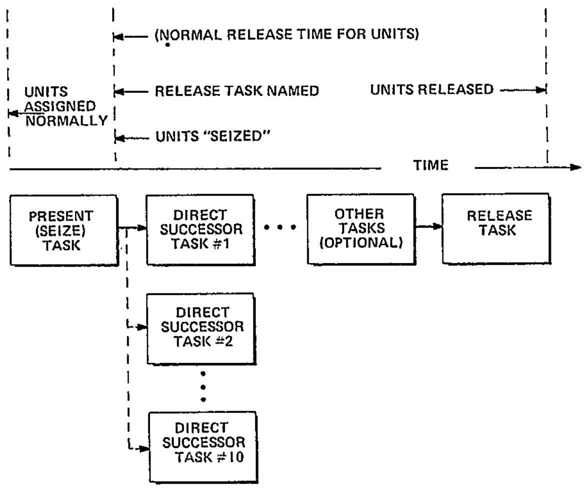


Figure 8. Unit Seize Operation

Type 6 - Register State*

A successor task can be selected by testing the state of a specified global register, as indicated in Figure 9. The contents of a named register, N, are compared to a specified 6-digit number named VALUE.

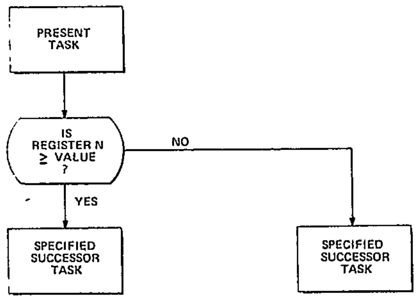


Figure 9. Register Test Successor Branch

Type 7 - Add to Specified Register*

The completion of the present task can cause a specified global register (N) to be cleared if desired (see Figure 10), and then to be incremented by a 5-digit number named DELTA. From one to nine direct successor tasks may or may not be assigned. If no successor task is assigned, the process ends when register N is incremented.

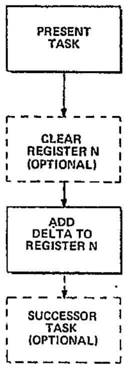


Figure 10. Register Increment Operation

*Successor types 6 and 7 enable interfacing of processes through the medium of communication with global registers.

Other Process Attributes

Process Starts

There are four types of process starts: single, multiple, derived, and embedded, each type being declared in a specific file. All may be processed concurrently, and each process may have many different starts of each type during the simulation run. Single process starts are those generated by the specification of one or more arbitrary, discrete start times. Multiple process starts are those spontaneously generated from a selected algorithm which generates process inter-arrival times. Such a continuing series of processes have "begin", "end", and "starting interval" specifications for controlling the behavior of the entire series. The derived process slaves the process start to a task completion.

Frequently a particular set of tasks appears repeatedly in a number of different processes. Such a set may be specified in BOSS as a separate process, and may then be called in the processes that use it as a subroutine or embedded process. As far as the calling, or parent, process is concerned, the sequence is a single task with its own task sequence number. In such a case, the embedded process start has the same NOW (time parameter for newest simulated time) as the task start.

Process Priority

A priority, based on a relative scale of priorities, is assigned to a process. The priority is employed by all the tasks of the process in competing for resources. The priority normally characterizes all tasks of the process. However, an optional feature enables the priority to be changed for individual, specific tasks in a given process. Provision has also been made to change the priority of embedded processes.

Default Conditions

In the Introduction, it was mentioned that BOSS incorporates certain biases and default conditions which greatly facilitate modeling and coding. This results from the pre-initialization of various codes and parameters to reflect the usual behavior of systems. In most cases this is effected by utilizing a zero-equals-blank convention in the file coding, and then making the zeroth selection code reflect the desired average behavior of the system. Therefore, the other code values are employed to take exception to the default case. By allowing a number of blanks, coding is reduced.

The major default conditions are:

1. FIFO-with-priority queue servicing of unit types
2. AND predecessor merging, enabling a task to start upon completion of one or more immediate predecessors
3. AND successor branching, enabling the current task to start zero, one, or more direct successors
4. Starting/terminating of processes with no predecessors/ successors specified
5. Specification of fictitious, or "dummy" tasks (those requiring no time or units) to effect successor branching.

Consider the following case: A task is started directly upon the completion of one predecessor; it seeks units and if unavailable, goes into a queue which is eventually serviced on a FIFO priority basis; upon completion it directly starts successor tasks. This very typical case can be effected with default coding.

BOSS Output Data Reports

The BOSS system is programmed to print out a wide range of system performance data. A full data report is spontaneously generated at the end of a run, unless the run is stopped by operator intervention. The end of a run may be a specified stop time, in simulated time units, or it may be caused by program operation, such as the exceeding of an array. Interim reports may be generated at regular intervals at the programmer's option. Such "snapshots" comprise a report, fully equivalent in volume to the final report.

Three major types of output data are generated:

1. Process performance (for each process type)
2. Unit utilization and queuing (for each unit type)
3. Chronology of events

Process performance data consists of the following:

1. Number of times process was completed and not completed
2 Mean time and standard deviation for completions and non-completions
3. Histogram showing cumulative distribution for completions and non-completions
4. Process threshold time statistics – an optional comparison of process time with specified time durations
5. Percentages of time process spent in queue.

Unit utilization and queuing data presented are:

1. For each unique unit, number of times used and percent of time busy
2. For each unit type (pool), percent of time queued, average length of queue, and maximum length of queue based on total time and interim time since last report.

Chronological data trace includes:

1. Process starts: time and mode of starting (derived, etc.)
2. Register updates showing time and initial and changed value of register.
3. Tagged events. This is a programmable option. Denoted as a numbered "tag", it is the completion of any task in any process. Tags are declared as global quantities in a particular file.

BOSS Mapping

The suggested flow chart convention for a task is shown by the annotated rectangle or block in Figure 11. These annotations describe the type of information represented on the flow chart and the files that the BOSS program has set up to contain this information.

The flow chart for a process consists principally of these rectangles (task blocks) connected by lines, and with symbols and notations added to the flow chart as needed. Information given above, in, and below the task block is discussed below. The compactness and utility of this notation greatly enhance the modeling process.

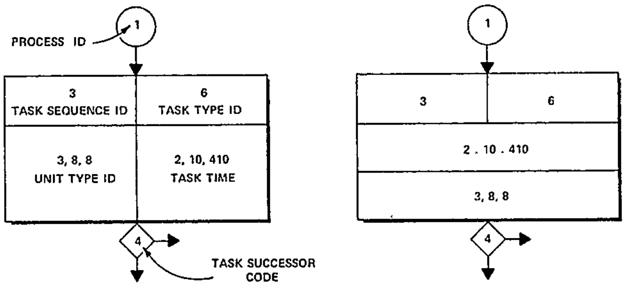


Figure 11. BOSS Block Symbology

Information Above the Task Block

The process TID number is given in a circle at the top of the flow chart. If required, a fan-in element in the form of an AND or OR gate symbol is shown upstream of the blck representing a task. AND corresponds to a predecessor code "0"; OR corresponds to a predecessor code "1." A fan-in element is normally not indicated for a task having one immediate predecessor; a default code of "0" is assumed.

Information In the Task Block

The typical blocks shown in Figure 11 represent two forms of notation. Each task sequence number within a process (local) is shown in the upper left portion of the pertinent task block. These task sequence numbers do not have to be assigned in numerical order, although it is usually convenient to do so.

The task TID number (global) is given in the upper right portion of the block.

The task execution time is specified in the middle section of the block or lower right and is in the form (distribution type) : (first parameter) : (second parameter).

The required unit TID entries are given in the lowest or lower left section of the block, in the form of a single unsigned TID which indicates that one unit of this type is to be assigned to this task, and is to be released when the task ends. For multiple assignments multiple entries are made.

Information Below the Task Block

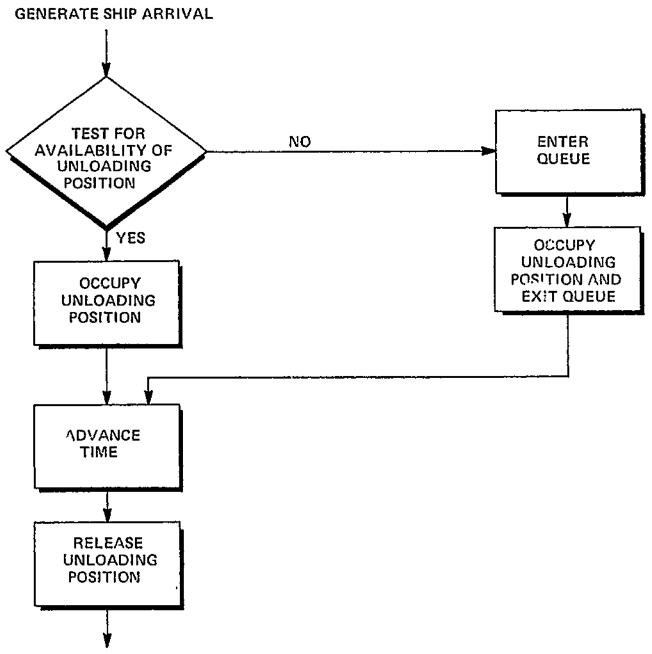
Any non-zero successor code is shown inside a diamond immediately downstream of the task. Fan-out lines to successors are given at or near the diamond symbol.

For the symbology example shown (Figure 11), the task is Task 3 of Process 1. It is a Task Type 6 which denotes that it requires the allocation of one unit of Type 3 and two units of Type 8 and requires service time selected from a uniform distribution with minimum value 10 time units, maximum value 410 time units. At the termination of the task, the flow branches conditional upon the unit availability for the next task (Successor Code 4).

BOSS Application Example

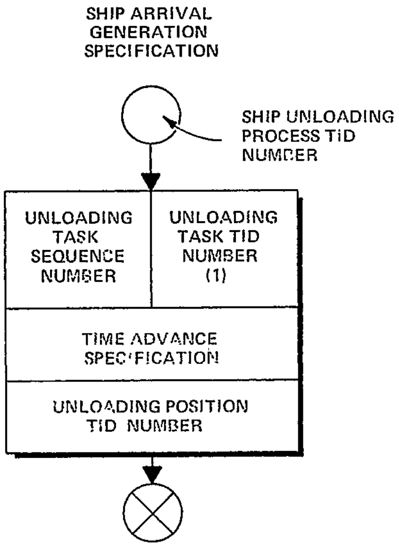
The following situation is to be implemented in BOSS.

Ships arrive at a port and undergo unloading, after which they depart the port. Arriving ships compete for unloading positions, and congestion may occur when all the available loading positions are filled. Consider all ships to be of one class (i.e., having common attributes), and that ships unable to acquire a position will queue for the position and eventually will be serviced FIFO. A process flow model of the operational sequence which describes the above situation is shown in Figure 12. To implement this model in BOSS, assignment of Type ID numbers is made to the process (ship unloading sequence) and the pool of unloading positions. A BOSS map (Figure 13) is then produced showing the above operations to be incorporated into one BOSS task. (This compactness is due of course to the assumptions encompassed in the BOSS program). The numeric codes employed on the BOSS diagram are then transferred to appropriate data card files for input to the program.



GENERATE SHIP ARRIVAL
TEST FOR AVAILABILITY OF UNLOADING POSITION
NO
ENTER QUEUE
OCCUPY UNLOADING POSITION AND EXIT QUEUE
YES
OCCUPY UNLOADING POSITION
ADVANCE TIME
RELEASE UNLOADING POSITION

Figure 12. Ship Servicing Process Logical Flow Diagram



SHIP ARRIVAL GENERATION SPECIFICATION
SHIP UNLOADING PROCESS TID NUMBER
UNLOADING TASK SEQUENCE NUMBER
UNLOADING TASK TID NUMBER (1)
TIME ADVANCE SPECIFICATION
UNLOADING POSITION TID NUMBER

Figure 13. BOSS Block Diagram for Ship Servicing Process

Conclusions

This paper has presented a narrative, introductory description of the BOSS simulator without formally presenting a detailed application with coding, matters best left to a detailed application course.

One of the major benefits which has been obtained from BOSS is a rapid turn-around-time medium for initial instruction in simulation. Normally one day of individual instruction or two days of class are sufficient to enable a competent analyst to begin independent simulation activity. Because of the compactness of coding and notation, and the various default conditions inherent in the program, application is also quite rapid and this enables a quick response for dynamic jobs such as are likely to occur in the marketing environment.

The fixed-field, numerical coding also enables inspection debugging of data base cards.

Upward compatibility over all program versions enables applications developed at any time to run on the latest version of BOSS.

BOSS has drawn criticism on its structure. Admittedly a certain amount of flexibility has been sacrificed to obtain the BOSS problem-oriented program configuration. However, the developers and appliers of BOSS feel that the increase of utility clearly outweighs the degradation of generality, because BOSS puts a useful, debugged tool into the hands of the person having a simulation problem, thus letting his address himself directly to modeling his system.

References

1. Meyerhoff, A.J.; Roth, P.F., Shafer, P.E. and Troy, J.P.; BOSS, Applications Manual, Revised Edition; July 1970? Burroughs Corporation, Defense Space and Special Systems Group.

2. Roth, P.F. and Meyerhoff, A.J.; "BOSS Simulation of a Time Sharing Message Processing System for Bank Applications." Proceedings of Third Annual Simulation Symposium, Tampa, Florida, January, 1970.

Acknowledgement

The author wishes to acknowledge the contribution of Albert J. Meyerhoff, Senior Staff Scientist, Burroughs Corporation, a co-developer of BOSS for his helpful critique and comments on this paper.