PROPOSAL FOR A NUCLEUS I/O SYSTEM

by

Robert F. Rosin

DAIMI PB-23

January 1974

Institute of Mathematics University of Aarhus

DEPARTMENT OF COMPUTER SCIENCE

Ny Munkegade - 8000 Aarhus C - Denmark

Phone 06-12 83 55

PROPOSAL FOR A NUCLEUS I/O SYSTEM

Abstract

Since the typical so-called microprogrammed system provides only hardware oriented input-output facilities, it is highly desirable to supplement this with a nucleus of software supported i/o routines.

A proposal is set forth for such a nucleus system to be implemented at the lowest level of a multi-level computer system.

Although the proposal is of a general nature, specific reference is also made to the Rikke-1 hardware being developed in the Computer Science Department at Aarhus University.

CONTENTS:

I. BACKGROUND AND PURPOSES

Most computer hardware systems seem to be designed from the bottom up. As a result the i/o facilities of most systems tend to be device oriented, require much concentration on timing considerations, and usually are not at all directly suited to the needs of users who tend to take a top-down approach to their applications. Therefore, almost all computer systems are equipped with a layer of software, which might be considered cosmetic in function, upon which all other software is built. For example, it is generally impossible for users, or even system programmers, to write code which directly exercises the i/o facilities of System/360 or System/370. These operation codes are executable only in "supervisor state", and non-supervisor programs access i/o only through a software interfac e which is specified as part of the operating system.

There is no reason not to expect the same to be true for almost any other system. In fact, however, there is another possibility brought about by the rather recent access of users to multi-level systems, where one level is usually called microprogrammed. It is quite possible to provide a microprogrammed implementation of the preferable user-oriented i/o system, which can be accedded by all user and system programs, such that the "hardware" appears to support a rather suitable (from the top-down point of view) set of input-output "instructions".

It is this kind of interface which is suggested in this report. In the paragraphs which follow the purposes and non-purposes for such a system are discussed. Later sections of the report are directed toward the basic concepts and organization of such a system, detailed descriptions of the proposed system, and finally some notes with respect to implementation strategies both in general and with respect to possible implementation on Rikke-I.

However, before going on to discuss the purpose of such a system in greater detail, it must be made very clear that this is merely a proposal. As such it cannot be accepted as it stands, but needs evaluation, criticism and modification. It is suggested that the most vital method for achieving these ends is through attempted use of the facilities proposed. Although a more abstract and philosophical evaluation will also be useful, nothing can replace the experience and evidence which can be gained from attempting to apply the suggested routines.

1.1 Purposes of the proposal.

The task undertaken has been to develop specifications for a low level input-output system which might be built for Rikke-I. It is clear that one must know what is needed, and for what purposes, before implementation is considered. Therefore, we have not concentrated particularly on matters of implementation or on the Rikke-I itself.

On the other hand, just as a proof is never developed in the neat way that it is finally expressed, so have these ideas been developed through discussions at many levels, one of which certainly included some thoughts about implementation in general, and another level which concentrated on the facilities of Rikke-I. Therefore, the final section of this proposal addresses some of these matters, but only in an incomplete manner.

It may be that the system finally specified in the last iteration of this proposal cannot be built on Rikke-I or on a number of other machines. From this it might be concluded that this reflects as much on those machines as it does on this proposal, or even more so. On the other hand, there is no evidence at this time that this system is incompatible with any given hardware system, nor is such a result expected. It may be, however, that implementation of the proposed nucleus on one system or another may demand more hardware resources, both storage and time, than a group of users may wish to relinquish to such a facility. Again, that is a matter of evaluation which goes far beyond the puposes of this proposal.

1.2 Purposes of the system.

We begin by listing a set of purposes to which the proposed system is not directed. It is not supposed to:

1. Provide a totally device independant system.
2. Provide a totally character-oriented system.
3. Provide a resources-management system which, for example, decides when a calling program should be put into a "wait" state.
4. Provide a monitor routine, let alone a multiprogramming supervisor.

Instead the purpose is to provide a set of facilities in which all of the above and more can be implemented in a manner far more convenient than if the nucleus system were not available. That is, rather than offering a finished system, the intention is to provide a set of tools which will allow the construction of a wide variety of potential systems, hopefully without any limitations not already imposed by the host hardware. In particular the system should:

1. Provide a uniform method of control of and access to i/o resources, including coordinated treatment of interrupts.
2. Provide a well defined software interface which can be relied upon to remain largely unmodified in spite of future hardware adjustments or implementation strategy decisions.
3. Provide a system much more closely matching the needs of the user than does the host hardware.
4. Offer access to all system resources at a low level when this might be useful; for example, when requested, but only when requested, to have control transferred to a particular location upon occurrence of an interrupt.
5. Provide a system which, at the very least, has the capability of supporting O-code, P-code, and Nova emulators, when and if these are fully implemented, on Rikke-1.

1.3 Historical footnote.

Many of the ideas contained in this proposal are based on similar concepts which appeared in a scheme adopted by the microprogramming and emulation research group at SUNY at Buffalo in early 1972. The principal author of that proposal was William C. Hopkins. Some of those ideas were, in turn, taken from the i/o system which was in use at the University of Michigan on the IBM 709 and 7090 computers between approximately 1960 and the time that the system was replaced by an IBM 360/67. That system was mainly the result of the efforts of Robert M. Graham.

Unfortunately (both for us and for them) the Buffalo group never had the opportunity to implement their proposal, so the final demonstration of its suitability to support, for example, the Nova i/o instruction set, was never made. On the other hand, they made considerable simulated use of the system and were quite convinced that it indeed met its goals.

II. CONCEPTS UNDERLYING THE SYSTEM:

This system can be described as being an alternative/generalized/low-level/block-transfer/word-oriented/interrupt-handling/multi-purpose input-output system. The paragraphs which follow expand on each of the phrases found in that sentence, and will help the reader in underatanding the motivation behind the system which is to be described later.

II.1 Alternative/ As stated earlier, most software systems include an input-output nucleus which, for the user, replaces the hardware input-output facilities of whatever machine he is using. It is necessary that the user not have the ability to exercise directly the i/o facilities of the machine he is using when such a system is in use, for to do so would surely result in a breakdown of the complicated coordination functions which the nucleus performs by removing needed information from the controlling data base. For example, if a user takes control of a particular interrupt without the system's knowledge, then the nucleus will probably find itself expecting an interrupt from some device even though that interrupt has already arisen and been processed outside of its domain. Any later attempt to use that device via the nucleus will show that it is "busy" with an interrupt expected.

Most contemporary single-level hardware systems allow for the isolation of the nucleus from users by putting i/o instructions in a special class; for example in the class which can only be executed when the machine is in a so-called supervisor state. In other systems, the integrity of the nucleus can only be enforced by the willingness of users to comply with a self-enforced set of conventions. In a two-level system such as Rikke-I, the integrity need only be voluntarily supported at the micro-code level, because the emulators residing in control store can and should guarantee correct use of the lower level system by routines at the higher level. That is, the emulators must observe the conventions voluntarily while they are enforced by the nature of the emulators themselves at the higher level(s).

For all intents and purposes, correct use of a nucleus is identical with a redefinition of the computer being used. The input-output instructions of the system, say Rikke-I, disappear, and their functions, or a related set of functions, are supplied instead by a set of routines which for all intents and purposes become elements of the hardware. Therefore, we might call this machine Rikke-I.5 and prepare ourselves to work with this alternative.

II.2 Generalized/ If the world of real systems is characterized by device-dependancy, then ideal systems would require pure and complete device independance. For example, any device should have a well defined interpretation of such functions as rewind, seek, and empty print buffer. Although this ideal is achieved to some extent in particular high-level, file-oriented systems, the system proposed here is intended to be realistic and cannot and will not pretend to offer such purity.

On the other hand generality suggests that the nucleus should deal with physical devices, but allow the user to consider them to be logical devices. There is no reason that those functions which are common to all devices, or to a subset of those devices attached to some system, cannot be initiated in exactly the same way, independant of device peculiarities. The user need not be concerned with the "handshaking" necessary to communicate with many terminals, or the fact that the buffer emptying function for two output devices differ significantly. By developing a standardized calling sequence and a standardized set of functions which the system recognizes and supports, the task of the user, which is hopefully of greater interest than to exercize a set of i/o devices, can be far more easily accomplished.

Furthermore, the nucleus can very effectively serve to provide interpretation for actions which the devices themselves may treat ambiguously, and protect against accidental transmission of messages which effectively paralyze the system; for example, transmission of a character whose definition may be to cause the printer to "go down."

Generality also implies some insulation from the timing-dependant nature of many typical i/o operations. This can be extended to include help for masking the timing characteristics of the several storages which typically are found in a multi-level system such as Rikke-I. For example, it became apparent during the development of this proposal that a nucleus could allow (if not support) the specification of a wide variety of data transfers not normally categorized as i/o. With some restrictions one can conceive of device-to-device and memory-to-memory transfers being included along with the more traditional memory-to-device operations.

The logical rather than physical nature of the devices seen by the user through the nucleus interface should certainly extend to naming. There seems to be no good reason for the particular physical connection between a device and the computer to determine its logical name, any more than the particular bit pattern that rewinds a tape should determine the name of the routine the user employs to initiate that function. Furthermore, logical naming allows aliases which, for example, can be used to provide two distinct names for the printer, depending upon which character graphics are available on the print chain currently mounted. This idea also can be extended into the realm of character- versus word-oriented transfers on such devices as tapes. This general topic is further treated in the appendix.

II. 3 Low-level/ A high-level system is designed with a particular set of applications and goals in mind, and more importantly with the idea that no other applications are to be considered. By "low-level system" is meant one which has at best a very broad set of applications in mind (see section II. 7 below) and the further hope that other applications, not presently being considered, will not be excluded by the design.

With respect to an input-output system this means that the nucleus should make as few decisions as possible and conceal the smallest amount of information manageable, while still being able to carry out its functions. Such information as the status of a requested i/o transfer, the availability of a particular device, and the number of words transferred before an error arose, must all be available. Furthermore, rather than putting a calling program into a wait state when a request cannot be met, the nucleus should reflect the appropriate information back to the caller, and then provide the optional capability for the caller himself to request that he be put into the wait state. Additional examples of this aspect of the design will appear throughout this proposal.

More generally, the nucleus is not to protect the user but to serve him. This implies, to take only one example, that not only are deadlocks between requests not prohibited, they are completely possible. The user bears the responsibility for their avoidance, as in any other low-level system

II.4 Block-transfer/ It is recognized that i/o operations typically take place one word at a time. This understanding becomes even clearer in the environment of a two-level system, in which the physical i/o buffers are quite visible to low level programs. On the other hand, there are many devices which operate most effectively, and sometimes only, when processing a block of information. For example, although a printer is reached through a one-word port, it actually stores characters in its own buffer until it is instructed to print; and a disc, as well as magnetic tape and higher speed punched paper tape devices, is almost invariably expected to process a block of information, even though the data must again pass through the one-word i/o port somewhere in the process.

Of course it is possible to program a low-level system in such a way that every transmission of a single word between the outside world and a portion of the computer's storage is made painfully obvious to the user. Indeed, the attention needed to manage such low-level resouces must be given, for there is no "magic" which makes this happen automatically. (The use of a channel for such purposes is merely the assignment of such tasks to a specially designed processor whose programs are traditionally unalterable, although the use of read/write control storage in channel processors seems to be changing these concepts as well. If one cares to do so, he may compare the nucleus system being proposed here with an integrated channel implemented in microcode.)

Finally it must be recognized that users see their programs as working on sets of data, messages, arrays, and other such items. It is only at the lowest level of program refinement that the idea of a computer word ever enters the picture, if even then. Therefore, a block-oriented system appears not to detract from what the user wants. Clearly a word-, character-, or even bit-oriented system can be implemented in software on top of one supporting block-transfer, as has been done numerous times in practice.

II.5 Word-oriented/ Once the decision to specify a block-transfer system has been taken, the nature of the information being transferred still must be considered. With respect to many devices there is no choice. Disk blocks contain words, and only words. But there are many other devices which appear to process characters only, and a larger number which seem to be able to handle either words or characters. Line printers expect characters to come across the interface, for example, and a paper tape reader can be thought of as transmitting either streams of characters or streams of words, in which many of the bits are zero.

The decision to choose a word-oriented nucleus is based on the fact that a word-oriented system can always be programmed to appear to be character-oriented, but the inverse is not usually possible. For example, a character-oriented system should recognize the end of message character on the punched paper tape reader and ignore all following characters, rather than transferring them to the computer. However, in this case it would be impossible to read programs in absolute binary form from that tape, because there would be no way of assuring that the end-of-message character would not appear as a binary pattern in some instruction. The possibility for including escape characters to overcome this limitation is, of course, recognized, but this serves quickly to destroy the basic character-oriented nature of such a system. The concept of a dual-mode system with a large, homogeneous character set is treated at some length in the Appendix and may be considered as an extension to this proposal.

II.6 Interrupt-handling/ It has been generally recognized that interrupt facilities can be used with great effectiveness in the support of input-output operations. Although at some level a polling system of sorts must exist, even if it is merely a hardware device dedicated to sensing the presence or absence of activity on some i/o line, even the lowest level computer-on-a-chip devices support interrupts in most cases. Most microprogrammed computers simulate interrupts at the target level, whether or not they exist at the lower level. In order to provide similar capability, the nucleus must provide interrupt capability to those routines calling on it for service, whether or not interrupts exist in hardware.

However, whereas some interrupt based systems always reflect such events up to the level of activity which initiates i/o, most systems allow the caller to specify "don't care" when initiating input/output operations. One application of such a capability is to send messages to an output printer and not bother to be interrupted when the message has been received and the device is again available. If the caller sends a message and the device is blocked, several alternatives might be considered, but, consistent with the low-level criterion, the blocked nature of the request is reflected back to the caller in this system rather than, for example, stacking the request in a queue. But this strategy requires that the system not only provide interrupts at the caller's level, but it must also have the capability of processing those interrupts if the caller specifies that he does not wish to handle them himself. In any case, the caller must be able to obtain the current status of any pending request, but this becomes especially important in the case where the caller desires to poll rather than accept interrupts.

II.7 Multi-purpose/ As implied in several of the preceding paragraphs, the nucleus should be able to provide the user with a wide variety of options as to its application, and more importantly limit as few options as possible. If one could guarantee that no possible application were disallowed, then the system could be called "general-purpose". However, no such claim is made here. On the other hand, there are not potential application areas which are purposely excluded in this proposal. It is more likely that limitations imposed by a particular implementation, especially when these reflect some basic limitations of the host hardware, will be more detrimental to certain applications than would be the basic nature of the nucleus proposed in this document.

There are several possible approaches to applying the facilities of the proposed nucleus. One would be merely to reflect each virtual i/o command down into the nucleus by the emulator and to reflect all interrupts and other responses back up to the emulated machine. Another approach would be to emulate as precisely as possible, but most likely without regard to timing, the i/o facilities of some particular computer, and include this capability within an emulator for that computer. It may also be possible to interject, in between nucleus and one or more emulators, a very complex multiprogrammed or multitask emulation facility, which partitions devices among a number of active emulators and/or users, maintains consistancy of all requests, and allocates system resources, such as storage and cpu time as well as i/o facilities, among them.

III. BASIC SYSTEM CONCEPTS

In this section we deal with the overall characteristics of the system being proposed, whereas in the following section specific proposed entries, parameter lists and values returned will be presented. There are three topics to be covered in the present section: devices, interrupts, and calling and return sequences.

III.1 Devices

III.1.1 Classes of devices

It is difficult to distinguish between input-output devices and typical "fast" storage in a multi-level system such as Rikke-I. For example, a wide store containing 64-bit words is connected to an input-output port as though it were four logical devices, one for each 16-bit field. Although there would be some temptation to treat this as a "fast" store rather than an i/o device, its presence on an i/o port requires that its use be coordinated with that of the other devices on the same port, lest the integrity of the nucleus be imperiled. Except for general purpose working registers and the other registers, in a multi-level system such as Rikke-I everything connected to it appears in one way or another to be an input-output device.

The result is that information can be transferred only between i/o devices and such registers, as is in fact the case with the strict hardware definition. However, most i/o requests in a typical single level system can be used to specify a transfer of information between what appear to be one i/o device and another in a multi-level system, a rather unconventional situation. A decision to support this latter form of transfer in a mult-level system leads naturally to the possibility of users expecting storage-to-storage transfers also to be supported, in all of the various possible combinations. This, in fact, is what the nucleus is able to offer with only a few implementation defined limitations.

However, no matter how clever or contrived, it is impossible to support this goal with complete generality. For instance, because of the wide differences between hardware limited word frequenceies for various devices, the nucleus cannot support transfers such as those between paper tape and disk. In a similar manner, although not logically impossible, nucleus support of storage-to-storage transfers at the same time that a device-to-storage transfer is taking place can result in some logical inconsistancies.

In order to compensate for these and other problems which arise in this general framework, devices are divided into classes based upon their speed and timing characteristics. In one class, called "fast" devices" (fd), are found the Rikke-I Working Registers A and B, Control Store, Main Store, and Wide Store, none of which have physical limitations on their use such that they must be read or written at a specified rate. Among the "slow devices" (sd) are found the Rikke-I input-output devices such as paper tape reader and punch, line printers, keyboard, and screen. Using this classification it can be stated that the nucleus is capable of supporting sd-fd transfers, as is expected. It should also be possible, however, to support fd-fd transfers, and perhaps some subset of sd-sd transfers.

It will be the case, however, that requested fd-fd transfers will be completed as soon as accepted by the nucleus, without returning control to the caller, which precludes the case of the calling program being able to process an interrupt which arises before completion of an fd-fd transfer. At the same time, there should be no restriction on supporting one fd-fd transfer simultaneously with one or more fd-sd transfers. Two fd-fd transfers cannot be supported at the same time, but such would be impossible to request given that fd-fd transfers are completed before the calling program regains control, in any case. (See also the next section in which interrupts are described in greater detail.)

III.1.2 Naming of devices

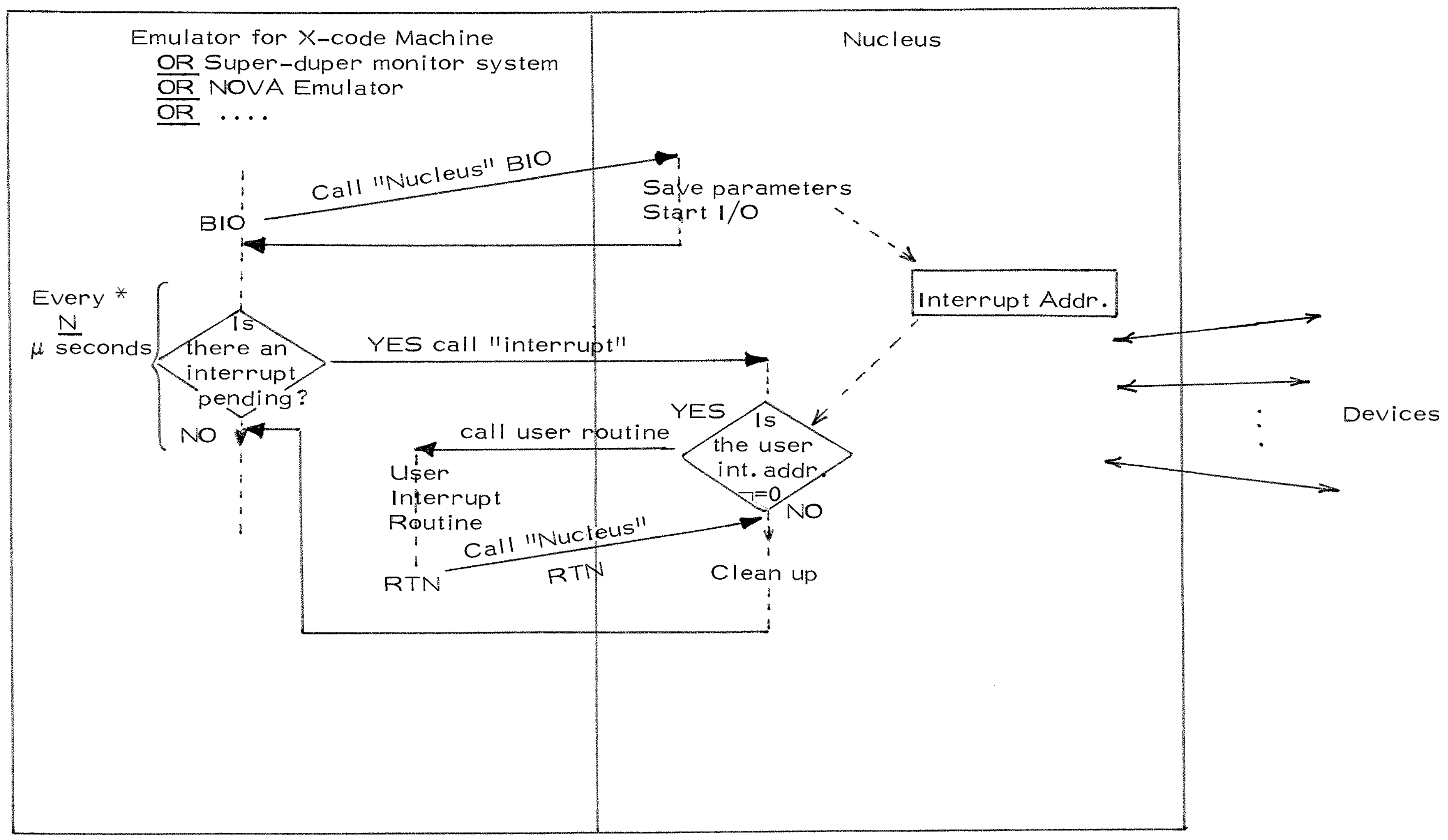
Each physical device has at least one logical name, and only that name will be accepted as a parameter by the nucleus. The user will be considered to have misused the nucleus if he somehow takes advantage of any hardware-dependant aspect of a particular logical name which might exist in some implementation; for example, the possible inclusion of the Rikke-I port number as part of the logical name of some device of that system.

For physical devices which have more than one mode of operation and/or alternative external representations of an internal logical character set, these variations will be supported by providing a distinct logical name for each separate set of characeristics recognized for a single physical device. "Device status" will clearly have to reflect the status of the physical device upon which a particular logical device is supported. Some other aspects of this facility are discussed in Appendix I as part of a broader treatment of the character set problem.

III.2 Interrupts

The concept of interrupt can be realized on both levels in a two-level system. At the lower level, the hardware either provides an interrupt facility, or one can be simulated directly at the hardware-level using polling. However, there is also a level of interrupt which is simulated at the lowest level, from the user point of view. Although it is the purpose of this proposal to provide a nucleus in which the user has to know as little as possible about the underlying hardware, certain problems of resource allocation make it useful to mix the two schemes.

Figure I is a schematic representation of the nucleus in relationship with a using environment. Note that the nucleus is not aware of, and cannot be expected to compensate for, any complex activities outside of its own responsibility. In particular, if a multiprogrammed system happens to be constructed using the facilities of this nucleus, this is completely invisible to the nucleus. There must be a scheduling and resource



* Note: there will be a Nucleus entry to determine if there is an interrupt pending.

Figure 1.

management facility residing in the left-hand (user) portion of the diagram, which is responsible for activitating i/o operations, reflecting interrupts to various users, etc. The nucleus serves merely to provide a convenient vehicle for addressing requests to and receiving responses from the hardware input-output devices.

In order to preserve the integrity of the nucleus, <u>all</u> hardware interrupts <u>must</u> be handled in the nucleus, and within a short time after their occurence. However, and here is where the system becomes mixed rather than pure in terms of level separation, in order to allow an emulator in a particular environment to preserve information particular to it and to release hardware resources which are agreed to be available to all low level programs (including the nucleus), hardware interrupts <u>can</u> be directed to a non-nucleus routine. However, in this case, the routine must enter the nucleus using the "interrupt" entry point within an implementation defined period of time after the interrupt has been detected in the hardware. The only difference, therefore, between the traditional approach and using user-detected interrupts is that the nucleus is informed of the interrupt a little later than it might be if this technique were not used.

Of course, this approach to interrupt handling assumes that the user follows exactly all conventions set down for safe and correct use of the nucleus, including timing and resource constraints. Of chief concern here is that the non-nucleus code must never assume the "meaning" of a particular interrupt nor can it invoke an i/o transfer using the nucleus in the event of an interrupt, but rather it must clean up its own affairs and call the "interrupt" entry of the nucleus as soon as possible. The advantages, in terms of smooth operation of the system and placement of few restrictions on normal use of the machine, are significant. It should be noted that on hardware systems where resources can be easily partitioned among several (levels of) uses, this mixed level technique can probably be avoided. But on a system such as Rikke-1, where use of an arithmetic unit may require preserving the contents of one or more registers, etc., it seems to offer a workable compromise between restricted use of resources by all users on the one hand and the lack of an i/o nucleus on the other hand.

In a system where the hardware does not provide interrupts, such as the early version of Rikke-I, the effect of interrupts must be provided through the use of a polling scheme. Every piece of low-level code, even within the nucleus itself, must sample the appropriate hardware flags in order to simulate interrupts at a frequency no less than that required by the nucleus, in order to maintain smooth and consistent flow of input-output information. Clearly this polling technique is completely consistent with the mixed level interrupt scheme discussed above, and can easily be replaced with a hardware interrupt facility when available.

Since the nucleus provides a pseudo-machine for support of input-output, it also defines an interrupt scheme for use by its callers. However, as can be seen from the diagram, hardware completion interrupts are reflected back up to the caller only after appropriate processing in the nucleus, and hardware word transmission interrupts are completely hidden from the calling level. As will be seen later, each invocation of the nucleus can be accompanied by a pair of user provided addresses specifying locations to be branched to in case of logical interrupt. (As noted, there may be many physical interrupts occuring before a logical interrupt is delivered to the user; most low level machines provide interrupts for each word transmitted to or from a device, whereas a block oriented nucleus system provides an interrupt only when an entire block of information has been processed.)

It will be seen later in closer detail that in every case, the nucleus requires that the routine receiving control after a logical interrupt re-enters the nucleus through the "rtn" entry in order to allow the nucleus to clean up its own affairs before returning to the user program at the point at which the interrupt occured. An alternative facility, called "remain" allows the user to specify that control should return to a point other than that where the calling program was logically interrupted. Similarly, the nucleus provides a "disable"-"enable" pair of entries to control the processing of logical interrupts.

The nucleus will not reflect up to user level an interrupt from a device which has not been requested for activation. All such interrupts will be lost. This implies that interactive typewriter terminals and graphics devices must be activated by a calling program through the nucleus, and an implementation with a large number of potential terminals will have to poll its inactive devices for signs of activity from time to time.

III.3 Calling and return sequences

The next section of this document contains specifications of entry points and paramenters. However, a few more general comments are in order here. It would be most desirable to define a nucleus calling sequence such that it could be used across a variety of particular implementations, but this is often a difficult and illusory task.

However, it can be noted that a careful and sophisticated use of a macro language and macro expansion can provide much of what would be ideally desired. For instance, it may or may not be the case that every nucleus entry named in the next section should be implemented as a separate physical entry point in some routine; for implementation purposes it may be best to concentrate them all into one entry with the specific name being passed as a parameter. (One instance where such an approach seems appropriate is in a three level system such as the QM-I, where, if each nucleus entry were a level-I instruction, this would lead to a large number of nano-words merely to support nucleus entry.) Similarly, it would be desireable, and certainly best from the point of view of system integrity and disciplined programming techniques, to allow access to dynamically available status information only indirectly through the nucleus. However, in most cases the cost becomes prohibitive, and the alternative of giving the user access to this information through direct access to the table where it resides can offer great economies. Again, the use of macro techniques can disguise such direct access, thereby offering a degree of protection otherwise lacking.

Therefore, in connection with the implementation of this proposed nucleus, it is suggested strongly that a sophisticated macro expansion system be provided to disguise and regularize its system dependancies. This technique has been used by many people in many similar instances, and is the one suggested by David Parnas in his discussion of modularization of programs.

Two other points are to be noted; first, the parameters passed to the i/o invocation routine reside in a table provided by the user. All status information, etc., provided by the nucleus will be updated in that table, and the user cannot expect to use that storage for any other purpose until the nucleus has returned control with a completion interrupt for that invocation. This approach is in keeping with the desire to provide a consistent and secure system, but at the same time it does not consume large amounts of hardware resources, such as storage, for tables. The calling routine is, of course, free to store other information relating to its processing of the invocation, provided that this information does not physically overlap any used by the nucleus. For instance, a multiprogrammed system built on top of the nucleus might put the identification of the user program requesting each i/o operation in an expanded version of the table. Again, a macro implementation could be very helpful in realising such a facility without requiring heavy dependancy on a particular format.

Secondly, it can be considered that this nucleus proposal is, in a way, a proposal for a co-routine rather than subroutine facility. In particular, note in Figure 1 that the transfer of control to a user routine for processing of logical interrupts is a call, with parameter passing, etc., and that it requires a "rtn" upon completion. The situation is, in fact, far more complicated. The figure does not show:

that more than one nucleus process may be active at any time,

that the user interrupt routine may call the nucleus and initiate other i/o activity before returning control to the nucleus, and

that information is passed from the nucleus to the caller upon every return from the nucleus.

IV. DETAILED DESCRIPTION

This section of the proposal contains a list of entry points, parameters, and values returned for every rountine in the nucleus. (The following section of the proposal discusses implementation-dependant aspects of the nucleus when considered as a possible software adjunct, with special attention paid to the Rikke-I hardware. Therefore, discussion of word width, table format, and particular encoding of various messages is not treated in this section.) The logical structure of the nucleus is our subject here.

There are four classes of entries to the nucleus:

Physical interrupt simulation
Logical interrupt processes
Initiation of transfer operations
Functional input-output operations (without information transfer)

These are treated in turn below.

IV.1 Physical interrupt simulation

"isinterrupt" - Although not necessary, depending upon the physical system upon which the nucleus is implemented, an entry is provided which returns a truth value indicating whether or not an interrupt is pending. This is particularly applicable in those cases where the hardware does not provide a physical interrupt mechanism, and where there is some advantage (time, space, and/or security) in having a single routine to convert a physical polling situation into a logical interrupt situation.

IV.2 Logical interrupt processes

"interrupt" - the entry to the physical interrupt handling routine in the nucleus. This must be called within an implementation specific amount of time after a physical interrupt has been detected in a user environment and/or after the user has polled and found an interrupt pending. In the case of a system which supports physical interrupts, the nucleus will provide a default routine which calls "interrupt" when an interrupt occurs. In every case, one parameter is passed: the location to which control is to be transferred upon completion of interrupt processing, normally the address of the instruction pending execution when the interrupt arose.

"rtn" - the entry used to return control to the location associated with interrupt currently being processed.

"remain" - same as return, except after re-entry to the nucleus, control is returned to the location following the call to "remain" rather than to the point of interrupt.

"disable" - disable the reflection of logical interrupts up to the user level until an "enable" command is given. (In fact, this this is an inhibit command, in that all interrupts occurring while "disable" is in effect are stacked rather than discarded, pending an "enable".) Upon entry to a user interrupt routine, interrupts are automatically disabled for one instruction, such that a call to "disable" can be processed without further interrupt activity.

"enable" - interrupts are enabled.

"wait" - control is given to the nucleus and not reutrned to the calling environment until an interrupt arises. If no interrupts are pending, then "wait" is equivalent to a dynamic stop command. ("wait" is also an option in the "bio" command, and will also usually be automatically invoked for fd-fd transfers.) Note that it is not necessary that user interrupts be pending for reasonable use of "wait". Rather, a nucleus request with the "wait" option can result in resumption of the user program at the instruction following that request upon completion of the operation.

(There is no provision in this system for interrupt priorites nor for device-specific disable and enable commands. These can clearly be implemented on top of the nucleus by providing a common handling routine for logical interrupts. On the other hand, if it is proven necessary, such a scheme can be implemented at the nucleus level.)

IV. 3 Function i/o (non-transfer) operations

"fio" - the calling sequence includes passing a code to specify the function requested, an identification for the device or transfer to which the function is to apply, and in some cases parameters containing additional information.

IV. 3.1 Functions of devices

These refer to logical devices, but will be reflected to physical devices where appropriate by the nucleus. There is one code which can be applied to any device, and then a number of device specific codes of which examples appear below. The universal code is "status", which results in the return of the following information with respect to the device:

not present
present but in use
present but inoperative
present and not in use

Device specific codes include:

rewind
backspace
clear screen
seek (where a third parameter would be used to specify
an address)

The use of these codes will result in the return of an information word which could contain any of the alternatives listed for "status" plus the alternative

invalid request

which might be offered in the case of an attempt to rewind a disk or backspace a paper tape reader or punch.

In the case of some device-specific operations, the time consumed might be so great as to suggest indicating their completion by logical interrupt rather than by waiting. In these instances a fourth parameter containing an interrupt address would also be required. (See discussion of interrupt address parameter under "bio" below.)

IV.3.2 Functions of Pending Requests

The parameters associated with these functions are the same as for device functions, except, of course, the functional codes which can be specified are different. The transfer to be affected must be specified, which (in the implementation scheme suggested in section III.3 of this proposal) can be the same as the storage address of the first word of the block containing the parameters for the transfer request involved. Although the cases in which information is requested from the nucleus could be satisfied by simple examination of the parameter table, in keeping with the spirit of preserving the integrity of the nucleus, this should be done <u>only</u> using these functions. A macro implementation, of course, would be consistent with supporting this approach without consuming execution time.

Three function codes are possible:

status
count
cancel

The first returns the following possible alternatives

complete/normal
complete/abnormal/error code
incomplete

The second function returns the number of words transferred so far in response to the "bio" request referenced. The third serves to cancel the effect of the request referenced, at the point which has been reached so far (but note that words already transferred cannot be deleted, and some devices require that only full records can be written), and to return the same possible messages as status.

Note that a function specified for a transfer applies to the current status of the parameter table for that transfer, and, in particular, if the transfer is complete, a "cancel" function has no effect, nor will the count change any longer. If the same area in storage is used for many successive transfers, as will normally be the case, then a status inquiry for an earlier transfer, which uses the same storage area as a later request, will result in the return of information reflecting the status of the request most recently initiated. There is no "invalid parameter" return for valid requests referring to areas of storage not normally holding nucleus parameter tables.

IV.4 Begin i/o operations

"bio" - This entry serves to request transfers of information using the services of the nucleus. Parameters are passed via a table, so that the only actual parameter for this nucleus entry is the address of the table in the calling program's address space. Once the request has been accepted by the nucleus, any change made to this table by the calling environment can result in chaotic behavior of the nucleus.

IV.4.1 Responses

"bio" returns a value to the calling program to indicate one of the following:

request accepted
source device unavailable
destination device unavailable
nucleus too busy to handle request
invalid device pair
transfer complete
invalid parameter

The first of these is the "normal" response indicating that the nucleus can meet the request with the resources available to it. The second and third responses ("unavailable") suggest that the caller may choose to use the device status "fio" to further identify the reason for the nucleus not accepting the request.

"nucleus too busy" is a function of the implementation. In some cases the nucleus may be implemented so that it handles only one request at a time, and in others it may be capable of considerable overlapping of transfers. In any case, use of the "wait" option (see below) precludes the return of this response, since the nucleus will retain control, processing interrupts, etc., until this request can be satisfied.

The fourth response indicates that the source and destination devices requested cannot be coupled together for a nucleus transfer. In some cases, such as sd-sd transfers, the response is mandatory, but the situation for other instances can be the result of an implementation dependancy rather than a physical impossibility. (see section II. I)

The "transfer complete" response will arise for accepted and processed fd-fd requests and for other accepted requests in which "wait" was specified. If the caller also specified an interrupt location for successful and/or unsuccessful completion, then control will already have transferred to that routine and back through the "rtn" entry of the nucleus before the "transfer complete" response is given for the request.

Finally, "invalid parameter" is used to indicate that some parameter, of unspecified nature, contains a value which is inconsistent or otherwise unacceptable to the nucleus.

IV. 4. 2 Parameter table entries

Although expressed somewhat abstractly below, the information to be passed to the nucleus will reside in fixed format in a table in the address space of the calling program. These parameters will, where appropriate, correspond in format and contents with the completion codes used for "fio" (section IV. 3. 2) and other corresponding purposes.

One facility, not previously mentioned, must be presented before going on to outline the "bio" parameters. In a block-oriented system it is always possible to specify a number of words to be transmitted which is fewer than the number in a physical block on some device, in which case the device-dependant aspects of the system define the results to be expected. For example, transfers from the device have no unexpected results, while transfers to such a device will generally result in zeros being written. However, this does not account for a large number of possible treatments of blocked devices in a general system, so an optional skip-before-transmit is provided in the nucleus. If the mode of transmission parameter is initially set to "no-transmit", then the number of words in the "no transmit count" parameter will be skipped before actual data are transferred. As in the case of incomplete blocks, transmission from such a device will behave as expected, while transmissions to such a device will be influenced by the conventions imposed by the implementation dependant-aspects of the nucleus. In any case, the ability to process incomplete physical records is useful in many applications of an input-output system.

The following entries are contained in the "bio" parameter list:

status - the same information returned by the "fio" status code request.

count - the same information returned by the "fio" count code request.

wait - a truth value specifying whether or not control should be returned to the user before completion of the request, if possible.

source device name

destination device name

source device beginning address (where appropriate)

destination device beginning address (where appropriate)

no-transmit-count - number of words to be skipped before transmission

transmit-count - number of words to be transmitted

successful completion interrupt address - if zero, then completion does not result in a logical interrupt

unsuccessful completion interrupt address - if zero, then completion does not result in a logical interrupt

The first two parameters can contain arbitrary information on entry, as can the device address fields where they do not apply. The status and count parameters will be set by the nucleus to reflect the current situation concerning the request before control is returned to the user, in the event that the request is accepted.

It is likely that each implementation will also require one or more words of additional space to hold request-dependant information during nucleus processing. In this way, the caller rather than the nucleus can control the amount and location of working storage needed for input-output processing.

V. IMPLEMENTATION NOTES

Two aspects of nucleus implementation appear in this section of the proposal. The first centers on general questions which have arisen in consideration of building an i/o nucleus, independant of any particular system, other than that itis based on a two-level organization. The second part of this discussion concerns some few specific suggestions and comments with respect to implementation of the nucleus on Rikke-I.

V. I General comments

V. I. I Devices

In implementing a scheme such as this, it is clear that the traditional structures of device control blocks (dcb) and request control blocks (rcb) will be needed. Note that device control blocks will be of two types, logical and physical, where each physical device may be represented at the user level by a number of logical devices; for example, one for character-mode input and one for word mode input from the same physical paper tape reader. Therefore, each logical dcb will contain a pointer to the dcb for the physical device it represents. An rcb will contain pointers to the dcb´s for the devices involved in the request. As mentioned earlier, it may make good sense to specify additional space in the user-provided parameter table so that it can also serve as the rcb after the nucleus has added whatever information is needed to initiate and maintain the transfer as requested.

Although not universally true, some devices can be bound simultaneously to more than one request, and provision must be made for treating these cases. For instance, main store can be used as both a source (say, to the printer) and destination (for example, from the CRT terminal) at the same time it is being used as source and destination in a storage-to-storage move. This is generally possible because of the wide disparity between transfer rates of various elements accessible to a system.

V. I. 2 Timing

In general, a large number of storage-to-storage word transfers can be carried out in the time it takes to move a single word, for example, from paper tape reader to storage. On the other hand, the very nature of fd-fd transfers, along with a desire to provide a system which is simple to use, leads to the decision to process all fd-fd transfers to completion before returning control to the caller. In a way this is not at all different from the stipulation made in most systems that interrupts can only arise between instructions. If that were not the case, then one would be faced with the perplexing question of how to recover from an interrupt. Clearly, more information than an instruction address must be saved, because one cannot be sure at which point in its definition the instruction should be resumed, based merely on a transfer of control.

This decision itself leads indirectly to the conclusion that sd-fd interrupts also cannot be reflected back to the user until any concurrent fd-fd request is satisfied. If this were not the case, then the interrupt routine would be free to request a second fd-fd transfer, which violates the premise just stated. Of course, careful use of values returned from "bio" (i. e., the one indicating nucleus busy) and the "wait" function could circumvent any potential deadlock, but the relative high speed of fd-fd devices implies that any delays realized by the strategy chosen will be minimal in any case.

In particular cases, where the data rate is within a small factor of the hardware instruction rate, it may be necessary to include rotational devices, such as disk and drum, in the "automatic wait" category in order to maintain their word transfer rates. However, this in no way alters the preceding conclusions, except to point out that the nucleus must be capable of maintaining careful control over all interrupt (or pseudo-interrupt) information, even if this is not reflected up to the calling program level.

V. I. 3 Strategy for Successful Implementation

Although programmers are often tempted to respond to a given challenge by stating how much time will be required to complete the entire job rather than commenting on feasibility, a more cautious approach is also more realistic and, in the long run, more successful. The nucleus provides a case in point. In order to have a system which can do something useful as soon as possible, and also a system which is reliable as soon as possible, the first step should be a measured one. Step-wise expansion is then possible along with the ability to provide fairly reliable estimates for the time and resources necessary to complete each successive step. The suggested plan is this:

first - implement a nucleus which handles only fd-sd transfers, limiting the range of possible device combinations, and providing an automatic "wait" for all requests. It may not run at the full desired speed, but the simplicity inherent in having no more than one process running in the nucleus may pay great dividends for early users.

second - remove the automatic "wait" restriction for those operations where timing appears to be easily managed.

third - extend the range of device combinations accepted by the nucleus, most generally in the area of adding to the spectrum of fast devices supported.

fourth - implement fd-fd transfers.

fifth - remove automatic "wait" in cases where the timing is more critical than in step two.

sixth - extend the range of slow devices by support of other than word-mode transfers. (See the appendix for suggestions for a character mode facility.)

V. 2 Implementation on Rikke-I

At the time of this writing, the Rikke-I documentation necessary to discuss the implementation of this proposal is not available. Therefore, it is difficult to offer more than very general comments. Of course, everything which is expressed in the preceding section applies in the case of Rikke-I. In particular, it is strongly suggested that a step-wise implementation be undertaken rather than one which attempts to accomplish too much too quickly, and results in too little being available too late.

The pseudo-interrupt technique will be mandatory until a satisfactory and correct interrupt facility is available on Rikke-I. The latter implies a masking facility, an event indicator which can be manipulated by a Rikke-I program, and an interrupt recovery facility which is logically consistent.

According the information supplied by Bjarne Stroustrup and Ole Sørensen, one can expect that an emulator running on Rikke-I should be able to sample for "interrupt" flags at least every 45 us, and give control to the nucleus within 1.5 us, if an interrupt has occurred. These timings, developed while they were developing on the 0-Code emulator, indicate that disk requests will always have to be completed before returning control to the caller, but that all other requests can use overlapped i/o and cpu processing.

Since no conventions have been agreed upon for calling sequences or similar software needs for Rikke-I, and since there have not yet been established any conventions for the state of Rikke-I registers when the machine is to be shared among processes, very little can be said with regard to this concerning the nucleus. However, since CS is not readable, it is clear that working data (control blocks in particular) will have to reside in MS or WA/WB, when i/o is in process. It appears that fd-fd transfers and those involving disk, which do not return control to the caller before request completion, will use WA/WB, while for all others most of the data can be kept in MS, with only intermittent use of WA/WB.

It has been suggested by some people that the nucleus itself should be coded in something other than the machine (or hopefully, assembly) language of Rikke-I, and in this way a large amount of CS, which would have to be devoted to the nucleus, could be replaced with code in MS to be interpreted by an emulator in CS. The savings to be gained, however, are illusory when one realizes that the emulator for the language in which the nucleus would be written would then have to reside in CS <u>at all times</u>, and this, combined with that portion of the nucleus which would of necessity be written in Rikke-I machine code, would probably prevent any other emulators from ever residing in CS.

VI. CONCLUSIONS

There are not many conclusions to be drawn from a proposal. Like any other proposed system, this nucleus would have to be tested by the most strict examination of all, construction and use, before one could render a final judgement. It is hoped that this happens.

When and if it can be concluded that there does exist a design for a nucleus which possesses the characteristics proposed in section II, then the next step is to investigate the building of such a system in hardware, and doing away forever with input-output facilities whose basis rests in hardware possibilities rather than software needs.

APPENDIX

Character mode input-output

A suggestion for circumventing the numerous problems regarding character sets is found in the OS series operating systems built by Stoy and Strachey (see Computer Journal, 1972). While their input-output philosophy is somewhat different from that expressed in this proposal, their approach to this one problem is not inconsistent with it.

Their suggestion is to implement to logical union rather than the logical intersection of the character sets available on the devices attached to the system. Even if one hoped that the ISO 7-bit character set could be the standard for some nucleus implementation, it must be recognized that there will exist graphics and control characters which do not lie in this set. The suggested strategy is to support an 8-bit character set which incorporates all of the various graphics and control characters found on <u>any</u> device attached to the system, rather than the usual approach of supporting only those characters found on <u>every</u> device in the system.

On input, each character would be translated into an 8-bit character within the nucleus standard (to be formulated by DAIMI). On output each 8-bit character would be converted into a 7- (or where needed a 6-) bit character for transmission to the addressed device. Some 8-bit characters might not have a function or representation on some device, and a device-dependant decision must be built into the nucleus for that situation. For example, if the NOT sign (¬) were not available on the printer, then it might be decided to represent it as _, or -, or ~ or blank or as an error reflected back to the system.

This implies the existence of a facility, somewhat like a translate table, for each device. However, by merely using the character value (i. e., numeric equivalent) to generate a word index and shift amount, it would be possible to determine <u>whether</u> conversion need be done for each character in a very few microinstructions, and using very few words of storage. If such a conversion were necessary, whether it be by simple substitution or by execution of a microroutine, then additional resources could be applied in those cases.

This scheme allows several very nice functions, especially since the user can assume a large number of characters is available to him, and that all devices have the same control characters.

There are two problems which remain in this context: the use of non-character devices and the use of characters which lie outside of this 8-bit character set. (Yes, the latter is quite possible.) For the former, there are two cases: where the device is purely word-oriented (e. g., disk) in which case no translation is ever done, and the case of a dual mode device such as a paper tape reader. In the latter case, a straightforward solution is to implement it as two logical devices, one word-oriented, and one character-oriented. In this way, the caller will determine by device name which facility he chooses, i. e., word-oriented or character-oriented transmission.

In the case of characters lying outside of the 256 supported, again two (or more) logical devices could be assumed. For example, it appears that chemistry-oriented characters sets, which involve characters outside of the 256 supported, can be obtained for a line printer. The printer with that print chain installed would merely have a different logical device name from the printer with a "standard" chain installed. It would be the user's responsibility to know that the 8-bit code which normally represents the symbol (will for that device only be used to print the chemists' symbol for a benzine ring. Internally the system would process it still as the (symbol.