A/S Regnecentralen
Copenhagen
February 1968

THE STRUCTURE OF THE RC 4000 MONITOR

Per Brinch Hansen

Abstract

A multiprogramming system for the RC 4000 is described. It allows parallel execution of programs on a non-swapping basis. Programs may initiate other programs in a hierarchal manner. Communication between internal and external processes is handled uniformly. The system allows for console conversation, interprogram communication, and file storage.

Contents:

Introduction

This report gives a brief description of a general-purpose multiprogramming system which is being developed by Regnecentralen for the RC 4000 computer.

The RC 4000 is a 24 bit, binary computer with typical operation times of 4 microseconds. It allows practically unlimited expansion of the core store and a standardized connection of all kinds of peripheral devices. Multiprogramming is assisted by concurrency of program execution and input/output operations, program interruption, and storage protection [1].

Our first experience with multiprogramming was a process control system implemented on the RC 4000 in 1967. This system performs time-sharing among well-behaved programs which remain permanently in the internal store [2]. L v

At the same time, Regnecentralen completed the Algol 4 compiler for the GIER computer. This uniprogramming system introduced standard procedures for the handling of common data files on various kinds of backing storage [3].

With this background we started the development of the present system. Our aim is to make time-sharing feasible on a machine with a minimum internal store of 16 k words backed by a fast drum or disk. Programs may be written in any of the available programming languages and may contain programming errors. The cooperation of programs is regulated by a monitor program with complete control over input/output, storage protection, and program interruption. The monitor provides the user programs with functions for the initiation and removal of programs, and for the communication between programs and peripheral devices.

Programs may communicate simultaneously with independent users through typewriter consoles. Programs may also communicate with one another. The community of programs share a set of data files on drum or disk.

The system has no built-in assumptions about program scheduling and resource allocation; it allows any program to initiate other programs in a hierarchal manner. Thus, the system provides a general frame for different scheduling strategies, such as batch processing, multiple console conversation, real-time scheduling, etc.

Objectives

The primary goal is to share a central processor and its peripheral devices among a number of programs loaded in the internal store. Automatic swapping of programs in and out of the internal store is not attempted. This means that the user will get immediate access to the machine provided there is room in the store and available peripheral devices.

We do not wish to impose any restrictions on individual programs with respect to their demand for storage, run time, and peripheral devices. Accordingly, it is taken for granted that certain programs will need most of the system resources for several hours. Such large programs must not, however, prevent other users from getting immediate access to the machine in order to execute more urgent programs of short duration. Thus, it is vital that the system permits temporary removal of a program in order to make its storage area and peripheral devices available for other programs.

Temporary removal only makes sense if it is possible to restart a program at a later stage. This requires reloading of the program into the store, as well as mounting and repositioning external documents. The need for repositioning also arises during program execution whenever the operator interferes with the operation of a peripheral device (by a mistake or in order to move a document to a more reliable device). Consequently, each input/output operation should be considered a potential restart candidate in the sense that is must be established that a document is still mounted in the correct position.

In order to minimize the manual handling of documents on paper tape or punched cards, the user should be able to retain files on drum or disk. The user should not be concerned with the actual location of a file on backing storage; thus, a file should be identified by means of a symbolic name. The user should have means of protecting his files against unintentional modification.

The system should support a conversational mode of operation which satisfies the following needs: any program should be able to open a conversation with any console and vice versa; a program should be able to receive messages simultaneously from a number of consoles; a console should be able to receive messages simultaneously from several programs.

At any moment several independent processes may be in progress in a multiprogramming system. This environment definitely discourages a mode of operation in which a main operator or a monitor program has to make all decisions about program scheduling and resource allocation. Thus, we consider it essential for the organization of the system, that the operator is able to delegate responsibility to other programs. This implies among other things that a program should be able to initiate other programs and communicate with these.

The design objectives may be summarized as follows:

1. The system will allow the parallel initiation, execution, and removal of programs in the internal store.
2. The system will allow the allocation of peripheral devices and the identification of documents mounted on them.
3. The system will allow the communication between programs, peripheral devices, and operators.
4. The system will allow the creation, updating, and removal of common data files.
5. The system will allow a hierachal control of the above mentioned scheduling and allocation functions. rc

We deliberately use the phrase will allow instead of will perform to stress that monitor functions are kept to a minimum to provide the most general frame for future operating systems. For example, if a program wants to remove another program temporarily, it calls a monitor function which stops the program and all its peripheral devices. The actual unloading of the program, hower, must be performed by the requesting program. Another example is that the monitor will inform a program when a peripheral device needs repositioning, but the actual repositioning is left to the program itself. ev

RC 4000

Multiprogramming is only feasible in a machine which allows the relocation, protection, and interruption of programs.

In the RC 4000, address modification includes indexing, indirect addressing, and relative addressing. This allows the initial loading of programs into any available storage area. Howyer, once started, a program will generate absolute addresses. This deficiency makes the segmentation and dynamic relocation of programs impractical. ev

The interruption system registers up to 24 signals which can be enabled and disabled individually. An enabled interrupt causes an immediate branching to a fixed monitor program with all interrupts disabled.

Storage protection is achieved by means of three additional bits in each storage word x). This protection key allows a division of the store into 8 distinct areas. The rules of protection within a running program is defined by a protection register of 8 bits. In store and jump operations, the protection key of the addressed word is used as an index to select a bit within the protection register. This bit determines whether the storage word is protected against the current program. Attempts to violate storage protection cause program interruption.

This system enables the monitor to change the protection situation when control is transferred from one program to another simply by loading the protection register. Another important consequence of this scheme is the possibility of establishing a hierarchy of protection among programs; that is, a program loaded with one protection key may be granted access to programs with other keys, and still be protected against these.

x) This is an extension of the original protection system described in ref. 1.

Further protection is achieved by privileged instructions that can only be executed within the monitor. These instructions include control of input/output, storage protection, and program interruption.

The protection system can be summarized as follows: a program interruption sets the machine in the monitor mode and starts a monitor program. In the monitor mode all instructions can be executed as long as they are protected. The machine leaves the monitor mode when the first unprotected instruction is executed. Program interruption will now result is the following is attempted: storing or jumping to a protected location, execution of a privileged instruction.

Processes

We shall use the word internal process to denote the execution of a sequence of interruptable instructions identified by a unique process name. The word program is reserved to denote a collection of instructions.

A distinction should be made between the names of programs and internal processes. An internal process may invoke a sequence of programs as in batch-processing. A situation may also exist in which several copies of the same program are executed in parallel. To allow identification of these independent processes they must be initiated with unique process names.

The word external process denotes the activities of a peripheral device identified by a unique process name. Some of the peripheral devices are merely interrupt sources, such as keys and clocks. Others are input/output devices on which documents may be mounted. A document is a specific roll of paper tape, a deck of punched cards, a printer form, a reel of magnetic tape, or a data area on backing storage. From the point of view of the internal processes, the name of an input/output process refers to the actities of a given document rather than to the identity of the device on which it is mounted (the identity of the device may in fact change if the operator decides to move the document to another device of the same kind). iv

The time-sharing and communication between internal and external processes is coordinated by a fixed program called the monitor. The monitor contains descriptions of all processes. It also contains procedures which perform privileged operations in an uninterruptable mode. We do not regard the monitor as an independent process, but rather as the implementation of indivisible operations which the internal and external processes can invoke by means of interrupts. This point will be illustrated in the sequel.

Process Communication

The parallel processes are cooperating in the sense that they can send messages to one another. This implies a mechanism for synchronizing two processes during the transfer of information. The requirement for synchronization is stated explicitly by the execution of a wait operation. This causes a delay of the process until another process executes a send operation.

Messages are transmitted in message buffers within the monitor. Buffers are assigned to processes upon request. They belong to the processes until they are released explicitly or the processes are removed.

The monitor administers a message queue for each process. Messages are added to this queue when they arrive from other processes. A process will serve its queue on a first-come first-served basis. After the processing of a message, the receiver returns an answer to the sender in the same buffer.

The internal processes use the following monitor procedures to communicate with other processes:

```
wait message(buffer, sender)
send message(buffer, message, receiver)
wait answer(buffer, result)
send answer(buffer, answer)
```

The procedure wait message delays the requesting process until a message arrives in its queue. When the process is allowed to proceed it is supplied with the address of the message buffer and the identity of the sender.

The procedure send message copies a message into a given buffer and delivers it in the queue of a named receiver. The receiver is started if it is waiting for a message.

The procedure wait answer delays the requesting process until an answer has been delivered in a given buffer. The result of the procedure specifies whether a normal or a dummy answer was received. A normal answer is delivered by a process in response to a received message. A dummy answer is generated by the monitor when a message is addressed to a non-existent process or to a process protected against messages from the sender.

The procedure send answer copies an answer into a buffer in which a message has been received. The sender of the message will be started if it is waiting for the answer.

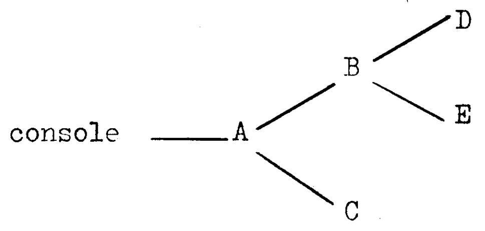
Internal Processes

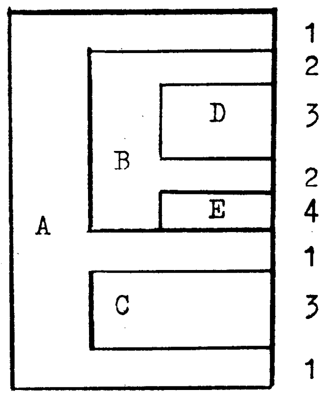
The internal store is divided between the monitor and the internal processes. An internal process occupies a contiguous storage area. It is either running (executing instructions, or ready to do so) or waiting (for an event outside the process). Scheduling among running processes is done on a round-robin basis.

Among the internal processes, the so-called permanent process has a special status. It remains in the store at all times. Its primary function is the creation and removal of other processes on request.

After initial system loading, the internal store contains only the monitor and the permanent process. The operator may now demand the creation, loading, and start of a parallel internal process by sending a message to the permanent process from a console. The operator supplies the process with a name, a storage size, a protection key, and a protection register.

Processes created by the operator may in turn spawn other processes within their own storage area. A family tree of processes and the corresponding storage allocation may look like the following picture (the integers illustrate a possible assignment of protection keys):





Each process is characterized by having a parent and a number of descendants. A parent has complete jurisdiction over its children. The parent may create, start, stop, and remove them by means of the following functions implemented in the permanent process:

```
create(process, storage area, protection key, protection register)
start(process)
stop(process)
remove(process)
```

The create function initializes a process description. The storage area assigned to the process must be within the parent's area. Also, the protection key and register must be a subset of those assigned to the parent. After creation, the process is in the waiting state. The loading of a program into the storage area can now be performed by the parent.

The start function sets the protection key of a process and changes its state to running. Also started are those descendants of the child which has been stopped previously by the parent.

The stop function places a process in the waiting state and waits for the completion of its input/output operations. The protection key is set back to its original value (i.e. the parent's key). Also stopped are those descendants of the child which are still running.

The remove function terminates the existence of a process and all its descendants. The message queues of the processes are emptied and dummy answers are returned to the senders. All peripheral devices assigned to the processes are released. Finally, the process names are removed from the system.

According to our philosophy, processes should have complete freedom to choose their own strategy for temporary removal and restart of their descendants. The system only supplies essential, primitive functions for starting and stopping of processes. However, at the operator's level one cannot avoid introducing specific means for loading, temporary removal, and restart of processes in order to get the system off the ground.

Sequential Input/Output

From the point of view of the internal processes, a sequential input/output process is created when the operator mounts a magnetic tape, a paper tape, a deck of punched cards, or a printer form on a peripheral device.

Magnetic tapes with standard labels are identified by the permanent process as soon as they are mounted on any available device.

All other kinds of documents are named by the operator from a console. This can also be done indirectly as follows: an internal process may ask the monitor for the assignment of an available device of a given kind. As answer the monitor delivers the present name of an available device. The internal process now tells the operator to mount a given document on the device without naming it.

Internal processes must be guaranteed exclusive access to sequential documents. Consequently, the system requires the explicit reservation of sequential documents.

An internal process initiates input/output by sending a message to an external process. The message defines a device operation and a storage area to be used as a data buffer. The monitor checks the validity of the call and connects the message to the queue of the device. If the device is idle, the monitor initiates the operation before returning to the requesting process.

The device signals the end of the operation by an interrupt. This causes the monitor to deliver a status word as an answer to the sender, and initiate the next operation in the queue.

Backing Store

The backing store is either a drum or a disk file. It is organized as a collection of named data areas. Each data area occupies a consecutive number of segments. A fixed part of the backing store is set aside for a catalog describing the names and locations of data areas.

Data areas are treated as external processes by the internal processes; input/output is initiated by sending a message to the data area specifying an input or output oper tion, a storage buffer, and a segment number within the area.

The identification of a data area requires a catalog search. The system tries to reduce the number of searches by creating internal process descriptions of active areas. When an internal process refers to a data area for the first time, the internal process is delayed while the permanent process performs a catalog search and creates a description of the area within the monitor.

The monitor has only room for a fixed number of area descriptions. When there are no more unused descriptions, old descriptions are overwritten by new descriptions. The victims will be those which have been left unused for the longest time. Internal area descriptions are also removed when the corresponding catalog entry is cancelled.

The system employs a protection scheme which is similar to the storage protection: each catalog entry is supplied with a protection key; the rules of protection within an internal process is defined by a simulated protection register set by the parent of the internal process.

An internal process creates a data area by sending a command of the following form to the permanent process:

new entry(name, size, protection key, parameters)

This causes the reservation of an area with the given name and size to be made and put into the catalog. The description may include a number of user-defined parameters.

In order to prevent the backing store from being filled with obsolete information, the concept of temporary areas are introduced. All areas are created as temporary areas. This implies two things: the creating process is guaranteed exclusive access to the area while it is processed; the area is automatically deleted when the internal process is removed. The deletion can be prevented by sending another command to the internal process.

Commands also exist for searching, locking, unlocking, changing, and removing ~~of~~ the description of an area. [illegible]

Console Communication

An operator opens a conversation from a console by depressing an interrupt key. This causes the monitor to select a message buffer and connect it to the console. The operator must now identify the internal process to which his message is addressed (after the first identification of a series of messages to the same process, he may just type an empty name consisting of a new-line character). Following this he can input a message consisting of one line which is delivered in the queue of the receiving process. If the operator depresses the interrupt key during input, the console responds with output of a new-line character in order that the operator may repeat the input.

The answer from an internal process to a console message merely has the function of releasing the message buffer.

An internal process opens a conversation with a console by sending a message to it. The message is connected to the console queue, and output is initiated if the console is idle. Before printing the message, the console identifies the internal process to the operator (this is suppressed after the first identification of a series of messages from the same process). The console completes the output independently by means of interrupts and delivers a status word as an answer to the internal process. It then proceeds to output the next message in the queue.

An internal process may also request that an output message is followed by an immediate answer from the operator. The answer consists of one line which is delivered in the original message buffer. If the operator delays the typing more than 10 seconds, an empty answer is delivered to make the console available for other messages.

We stress that a console is not assigned exclusively to a single internal process. Any internal process may communicate with any console and vice versa (provided the sender knows the name of the reciver). An operator may change the name of a console by sending a message from the console to the ~~internal~~ process. permanent

Real-time Synchronization

Internal processes communicate with a real-time clock by sending messages to it. After the elapsing of a time interval specified in the message, the clock returns an answer to the requesting process.

A similar communication is possible with interrupt keys; in this case, the interval is considered elapsed when the operator depressses the key.

Conclusion

A realistic evaluation of the system presented here can only be made after it has been used for some time. However, at the present stage certain conclusions can be drawn.

The uniform treatment of internal and external processes and the hierarchy of internal processes appear to be very general concepts.

A major weakness seems to be the inability of the monitor to delegate the responsibility for input/output operations. Thus, it is not only burdened with the assignment of devices, but also with checking the validity of all input/output operations. As a consequence, the machine will spend a large portion of its time in the uninterruptable mode where it is insensitive to external requests.

An effective solution to this problem would require a hardware protection of peripheral devices similar to the present storage protection.

Acknowledgements

The system presented here represents the thoughts and efforts of Jørn Jensen, Peter Kraft, Søren Lauesen, and the author.

References:

1. P. Brinch Hansen, The Logical Structure of the RC 4000 Computer, Bit 7 (1967), 191 - 199.
2. P. Brinch Hansen, The RC 4000 Real-time Control System at Pulawy, Bit 7 (1967), 279 - 288.
3. P. Naur, Features of the Gier Algol 4 System, A/S Regnecentralen, Copenhagen, 1967.