



HAL
open science

What is an Operating System? A historical investigation (1954–1964)

Maarten Bullynck

► **To cite this version:**

Maarten Bullynck. What is an Operating System? A historical investigation (1954–1964). Reflections on Programming Systems. Historical and Philosophical Aspects, 2019. halshs-01541602v2

HAL Id: halshs-01541602

<https://shs.hal.science/halshs-01541602v2>

Submitted on 30 Jan 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

What is an Operating System? A historical investigation (1954–1964)

Maarten Bullynck

Today, we could hardly imagine using a computer without an operating system, it shapes and frames how we access the computer and its peripherals and supports our interaction with it throughout. But when the first computers were developed after World War II there was no such thing. In fact, only about a decade after the birth of digital computing did the first attempts at some kind of operating systems appear. It took another decade before the idea became widely accepted and most computers would be rented out or sold with an operating system. With the development of ambitious operating systems during the mid 1960s, such as OS/360 for the IBM machines or Multics for an integrated time-sharing system, a more systematic framework was formulated that has determined our modern view of the operating system. Especially the emergence of time-sharing systems has traditionally been seen as a turning point in the development of operating systems. In the history of computing this has become a classic point of passage because of the sometimes fierce discussions between the proponents of time-sharing and the defenders of batch-processing in the late 1960s. Important as these discussions were for thinking about the use and about the users of the computer, as well as for the evolution of the software industry, the emphasis on this transition has biased the view on early computer systems.

As a matter of fact, the period between roughly 1954 and 1964 cannot be merely discounted as “empirical” or “prehistoric”, nor as the time of batch-processing systems. Rather a variety of systems were developed and the very idea(s) of an operating system had to be created from scratch. The neat classic storyline that goes from no operating system over batch processing system to modern multiprogramming or time-sharing systems¹ hides both

Maarten Bullynck, Université Paris 8, e-mail: maarten.bullynck@univ-paris8.fr

¹ This storyline captures only one (important) line of development and can be found in, e.g., [20, pp. 96-101], [64, pp. 6-18] or [43, 42]. Though also [16] follows this chronology, this presentation brings out that there were many systems and philosophies developing in parallel.

the variety and complexity of early systems and the fact that the notion of ‘operating system’ still had to stabilize.

This paper is based upon an extensive and systematic study of early programming and operating systems between 1954 and 1964 stopping just before commercial time-sharing set off.² The results are presented in three sections. They do not proceed chronologically, but go from the general to the specific, each section adding more layering to the classic storyline, bringing parallel developments into focus, and zooming in on the very question, what is an operating system? A first section describes the general background for the development of operating systems, viz. the evolution of hardware and software that made operating systems necessary and worthwhile. A second section proposes a panorama of early systems, classifying them into five categories: batch-processing; integrated systems; special-purpose systems; second-generation systems; and experimental time-sharing systems. A final section tracks the emergence of the term ‘operating system’ within the IBM community.

1 General Context: Operating systems and evolutions in hardware & software

1.1 *Beginnings in the mid 1950s*

The very idea of relegating part of the control of programming to the computer itself is actually born with the digital general-purpose computer. It is an aspect of what is commonly called the stored-program concept. Since the computer calculates much faster than any human being, the program should control the calculation while calculating. A logical next step would be that a program would control other programs. But in the early years, this control got no further than rather simple preparatory routines or bootstrapping routines. From the mid-1950s onwards, as the machines, the programming and the users evolved, this changed.

New memory technologies became viable, both working and storage memories. In the early 1950s, cheap magnetic drums were a good option to expand the capacity of working memory that was directly addressable by the computing unit. With time, the more expensive but faster ferrite core magnetic memories developed at M.I.T. would overtake them. As for the external storage media, the introduction of magnetic tape instead of punched cards or paper tape was essential for the development of operating systems.³ While

² The details of this systematic study cannot be included in this paper but will probably be published in book form with Lonely Scholar.

³ Magnetic tapes were introduced as early as 1951 on the UNIVAC computer, but did not become common for other systems until the mid-1950s. It should also be

the fastest punched-card readers of the 1950s could read upto 250 cards per minute, the tape systems trumped this by reading upto 15000 characters per second. This equals approximately 11250 cards per minute, which is 45 times faster than the card reader [32, p. 291]. It allowed for larger programs to be read into memory, and magnetic tape (sometimes also external magnetic drums) provided a way for easier and faster access to a library of routines. Of course, this acces was not random-access. Due to the physical qualities of the memory media, this access was either serial/sequential (magnetic tape), or cyclic (magnetic drum).

Another technological evolution was the introduction of buffer memory for the communication between input and output devices and the central processor, typical of such computers as the IBM 701, IBM 704 or the ERA 1103. Before that time, a number of strategies had been used to use the computing unit and its input and output peripherals synchronously, among them ‘spooling’ (putting information on tape rather than cards for speeding up I/O communications), ‘cycle-stealing’ (beginning an operation when the last one is not yet finished), read-write interlocks and, for large systems, ‘moonlighting’, using a smaller or slower computer as the I/O buffer to a larger or faster computer (a typical installation would involve an IBM 1401 and an IBM 7090). “With the advent of this phase [I/O buffer memory], input-output was taken out of the domain of mechanical speeds and placed in the domain of electronic speeds.” [9] Later still, because of the increasing demands from multiprogram design, special I/O channels with their own selector and multiplexing switches would be developed. Once low-cost, semiconductor-based processors became available, I/O channels would acquire their own processors instead of switches. One of the first examples of a processor-powered I/O channel was the Exchange system developed for IBM’s Stretch computer (1957).

The expansion of rapid storage for programmed routines goes hand in hand with the development of software. The latter half of the 1950s is traditionally seen as the years software development took off [20, pp. 79-108]. This is witnessed by the foundation of computer user groups such as SHARE for IBM users or USE for scientific UNIVAC users (both in 1955). These organisations regularly organized meetings to share programs and to exchange information on programming practices [4]. The same period also sees the birth of the first software companies such as System Development Corporation (SDC, 1957) that grew out of RAND’s involvement with the SAGE project, or Computer Sciences Corporation (CSC, 1959) etc. [18, pp. 29-56] In this same context, the first big programming systems were developed, some of which can, in retrospect, be called operating systems. One of the influential first systems was the Comprehensive System of Service Routines (CSSR) developed at MIT’s Lincoln Lab for its Whirlwind computer. This system would later be the starting point for the SDC’s programming system for project SAGE.

remarked that punched cards and paper tape remained in use, mostly in parallel with magnetic tape.

Other important systems, this time for linking and monitoring sequences of programs, were developed within the SHARE community and would lay the foundations for the batch processing system, typical of many commercial and scientific IBM installations of the 1960s.

Two punctual innovations, one in hardware and one in software, would prove to be pivotal for the further evolution of operating systems. First, in 1956, the interrupt was introduced for the ERA 1103A (sometimes also called Scientific UNIVAC 1103), a device that could interrupt machine operation to communicate with the processor.⁴ The hardware interrupt could be used to automate many of the manual interrupts that had to be handled by the human operator. It made more intricate monitor systems possible and was essential for developing multiprogramming (and later time-sharing) systems. In a multiprogramming environment, more than one program could be kept in memory, and while one program had to wait for input/output, another program could be executed. In its turn, the unfolding of multiprogramming would stimulate the design of more intricate hardware interrupts featuring trap mechanisms.

Second, from 1955 to 1957 an IBM team had been working on a scientific programming language that would eventually be called FORTRAN. As the first fully developed programming language⁵ and winged by IBM's dominance in the computer market, FORTRAN, and later FORTRAN II, quickly became popular and evolved into a must-have for most computer installations. The appearance of FORTRAN initiated much programming work on existing operating systems. They wanted to expand their functionality to include and accommodate the FORTRAN programming language. NAA's Fortran Monitor System (FMS, 1959), Bell Labs' BESYS-3 (1960), the University of Michigan's UMES (1959) or the RAND-SHARE operating system (1962) were all developed to get FORTRAN in an already existing systems.

1.2 Changes in the mid 1960s

The years between 1962 and 1964 mark a turning point, closing off a first phase in the development of operating systems. The emergence of time-sharing systems stand for this turning point, and at the horizon are the 'big' operating system projects OS/360 and Multics. However, these are rather the most conspicuous representatives of a broader and more general evolution. This evolution consists on the one hand of a gradual development of 'multiprogramming', and on the other hand the introduction of new and

⁴ There are earlier (or contemporary) instances of an interrupt, in special projects such as the DYSEAC, the SAGE system or IBM's Project Stretch, but its introduction on the ERA 1103A was the first 'commercial' appearance.

⁵ For the languages preceding FORTRAN, see [41].

faster memory devices. Together they made more complex and more flexible systems possible.

Multiprogramming breaks with the sequential processing and is in essence the idea that more than one program is running at the same time. In practice, this synchronicity of programs is only virtual. In reality, one program is executed by the main processor and that others are waiting or have been interrupted in the meanwhile, although I/O processing can happen synchronously with a program being executed. The hardware interrupt made the first instances of multiprogramming possible and the introduction of I/O buffer memory made it proliferate in many directions. This development, pushed onwards by the parallel development of software multiprogramming systems, called in its turn for some hardware innovations. It made scheduling of programs a necessity, as well as memory protection and a programmable clock. In a way, the idea of time-sharing a computer, viz. many users executing programs and using resources at the same time⁶, can be considered as an extreme form of multiprogramming.

Although multiprogramming profoundly changed the structure of computer systems, the transition from sequential to random storage media is easily the biggest game changer for implementing operating systems in the 1960s. IBM's 350 disk for the RAMAC (1956) was the first such random-access memory device, though it were rather the IBM 1405 and the IBM 1301 disk (1961-1962) developed to be used on the IBM 1410 and the IBM 7000-line of computers that revolutionized operating system design. Compared to the contemporary Hypertape-systems, also developed by IBM and operating at 170,000 characters per second, the 1301 disk drive is not only a factor faster, reading 112,000 characters per operation (with about 5 to 7 operations per second), but most of all, it has the same access time (about 0.150 per operation) for every section of data. The disk drives made it possible to leave the sequence-based logics of tape drives and drum memories and speed up the transfers between working memory and storage memory. This empowered software systems by reducing backlog and waiting times while enabling swapping between working and storage memory.

But there is more. By 1962-1964 it seemed that about every computer manufacturer had caught on to the idea of an operating system and had developed one.⁷ Before 1960 most development happened by the users of computer systems or had been done in research contexts (mostly funded by the military). Now the manufacturers started investing in programming teams that should develop the proper programmings tools to go with their machines. These included routine libraries, (macro)assemblers, compilers, loaders, programming languages, debuggings aids, but also master routines and operating systems.

⁶ As a General Electric's advertisement from the 1960s remarked correctly, "time-sharing is actually computer sharing."

⁷ It is also in the early 1960s that the first overview articles on operating systems appear: [55, pp. 290-294] and [46].

Looking at some of the major computer manufacturers⁸, they all came out with an operating system between 1962 and 1965 (see Table 1⁹). Some of these systems are rather primitive (GE's BRIDGE), others are rather classic batch systems (Philco's BKS or CDC's Scope), but most feature advanced multiprogramming next to batch processing. Time-sharing was not featured. Before 1966 it was still only developed in research settings, in commercial installations time-sharing had to wait until the late 1960s when IBM, GE, DEC, SDS and others would incorporate it into their operating systems (See section 2.6 for more details).

Manufacturer	Computer	Year	Operating System
Honeywell	H800	1961	Executive Monitor
Univac	Univac 1107	1962	EXEC I
Burroughs	D825	1962	AOSP
Burroughs	B5000	1962	Master Control Program
Philco	Philco-2000	1962	SYS; BKS
GE	GE-215/225/235	1962	BRIDGE
IBM	IBM 7090/7094	1962	IBSYS
Bendix	G-20	1962	EXECUTIVE
CDC	CDC 1604	1962	CO-OP monitor system
CDC	CDC 3600	1963	SCOPE monitor system
Honeywell	Honeywell 1800	1963	ADMIRAL Master Monitor
GE	GE 625-635	1964	Comprehensive Operating Supervisor
RCA	RCA 3301	1964	Realcom system
SDS	SDS 9000	1964	MONARCH
DEC	PDP-6	1964	Supervisory Control Program

Table 1 An overview of the first operating systems offered by U.S. computer manufacturers 1960-1964

The evolution from user to company is apparent in IBM's involvement. The first operating systems on IBM machines were developed by the (corporate) users such as General Motors (GM), North American Aviation (NAA), Bell Labs, Michigan University, MIT etc. They relied on input from the SHARE community of IBM users, but did not receive any direct support from IBM. Gradually, IBM as a company got involved too. They lent a hand in developing the Share Operating System (1959) that originated in the SHARE community. Later, IBM also slowly integrated NAA's FORTAN Monitor System (FMS) as a part of their 709/7090 FORTRAN programming system (1960-1962) [44, p. 818-819]. From this point on, they then started producing their own operating systems, first IBSYS (from 1962 onwards), later OS/360 (1965

⁸ We did not include information on computers that were not made in the U.S., but the same timeframe seems to be valid. For U.K. computers, e.g., the first operating systems appear in the beginning of the 1960s for the LEO III (1961) or Ferranti's Atlas and ORION computers (1962).

⁹ Two systems in this table are still the product of user development: the BKS system was developed by the Bettis-Knoll power plant; CDC's CO-OP system was the result of the efforts of its user group CO-OP.

onwards). In parallel, user-driven developments of systems slowly waned, though users would tinker around with the manufacturer's operating system to customise them.¹⁰

2 What is an operating system? A tentative taxonomy of early systems.

2.1 Automatic programming systems and operating systems

By using the very term 'operating system' one already, implicitly, subscribes to the philosophy that an operating system handles and partially automates the operation of the computer and in this sense replaces parts of the human operator's job. In particular, the manual operations that had to be executed on a 'control panel', a 'monitor panel' or a 'supervisory panel' were partially automated through the 'operating system'. On these panels the operator could handle the stops (after execution of a program or after a peripheral had stopped its operation), the interrupts (when a program or a peripheral malfunctioned or could not execute the command) and act on other signals. The 'operating system', by the philosophy implicated by its name, provided automated responses to these stops and interrupts so that not one program could be run between two stops, but a 'batch' of programs could run without interruption, hence the name 'batch-processing' for the first generation of operating systems. It helped to reduce idle time of the computer and to speed up the loading of programs. It also avoided some human errors and standardized loading and translating processes. In this context, it is often said that the operating system does the 'housekeeping operations'.

However, not only the operator was subject to automation, the programmer was too. The 1950s saw much work being done on automatic programming systems. This included the development of programming languages and compilers culminating in the creation of FORTRAN and, later, COBOL and ALGOL. But this included also engineering many other programming tools such as assemblers and macro-assemblers, interpretative routines, libraries of (sub)routines and utility programs to do the input-output communications, the conversion between decimal and binary (or other codes), etc. Automating the repetitive and dreary tasks of conversion, of searching, sorting and rewinding, of loading and linking with subroutines, etc. was as much on the agenda as simplifying the writing of programs.

¹⁰ Nearly all operating systems would be customised upto a certain extent. An example of a extensive customisation is Thomson-Ramo-Woolridge's version of IBSYS in 1962 [51].

Because all these things fall under automation, it is often hard before 1962-64 (and even today) to differentiate clearly between an operating system and a programming system. For instance, when W.L. Frank described a ‘program library’ in 1956, it included as a subset a number of “supervisory (or service) routines”. Those included: assembly and compiling routines; bootstrap and read-in routines; code checking and diagnostic routines; post mortem and monitoring routines; special arithmetic routines (floating point, complex numbers, double precision) [29, p. 6]. While bootstrap and read-in, as well as the post mortem and monitoring routines are clearly within the confines of ‘operating system’ nowadays, the other routines would rather file under ‘programming system’. Indeed, from one perspective, the operating routines are just part of the routine library in the programming system. This explains why, certainly before the mid-1960s, books and articles describing aspects of what we now call ‘operating systems’ often rather talk of ‘programming system’.¹¹ However, from another perspective, the operating routines oversee and control the programming systems, therefore they hierarchically are above the programming system. This latter perspective is an effect of the ‘monitor’ or ‘supervisor’-idea, viz. the automation of the operator, as present in the ‘operating system’-concept. This will be developed in more detail in Section 3 where the origins and evolution of the term ‘operating system’ will be tracked.

In what follows, a tentative taxonomy is given of early automatic programming systems that, a posteriori, may qualify as operating systems. Five main categories are distinguished: batch-processing systems; integrated systems; special-purpose and real-time systems; second-generation systems; and the early, experimental time-sharing systems.

2.2 *Batch-processing systems*

The concept of a batch-processing systems seems to have been born in the field of commercial data-processing and accounting. Because the tasks in business computing and accounting are more repetitive than in scientific computing, the idea of automating the operation of the machine in parallel with a streamlining of its use seemed promising. The idea of automating the (human) operator is an important one in the history of operating systems, and a central one for the traditional storyline. The philosophy of replacing the human operator by a program was most explicitly voiced by Bruse Moncreiff of the Prudential Insurance Company (but then working at RAND). In 1955 he wrote to C.W. Adams:

I have turned my attention to the problem of the day-to-day operation of an automatic data processor. The things that annoy programmers the most are

¹¹ See, e.g., the classic book by Rosen [57], but also [55], [28] or [35].

operators, so I am attempting to all but program him out of existence. There are certain phases of his work, mostly involving manual dexterity, which of necessity have been preserved. I have tried to remove all the thinking from his job, since this is what people do least efficiently. I like to think of this proposed routine as an automatic supervisor rather than operator since it will be telling the human operator what to do. (quoted after [2, p. 78])

In his article for the IRE Transactions, “An Automatic Supervisor for the IBM 702” (1956), he addressed the problem of running a commercial large-scale computing facility where “efficient day-after-day operation of the same routines” is needed. Since the “human operator cannot compete in speed with the machine in making routine decisions and in controlling the processing operations” the most efficient solution according to Moncreiff was “a supervisory routine [...] to keep the machine running efficiently in spite of the slowness and fallibility of the human operator.” [50, p. 21] But, as he noted, we first have “to gain a feeling for the complexity of a problem which as far as is known, has not heretofore been extensively investigated.” [50]

Around the same time, Owen Mock at North American Aviation (NAA) had been programming his 701 Monitor. The computer system at NAA had been housed in the Tabulating section of the accounting department making not only the historical continuity between accounting machines and digital computers, but also the continuance of everyday practices in this field tangible. A consequence of introducing Mock’s Monitor on the IBM 701 was a separation of tasks [49, p. 793]:

the separation of operations from users had some far-reaching effects. Among other things, it resulted in a philosophy that users should not touch the machine; in fact, they were not even supposed to touch their punched card decks. It also resulted in a philosophy of strict accountability [...] The end result was a user regimentation that, at first blush, would appear to run counter to the freedom mentioned in the introduction but which facilitated the introduction of operating system concepts

Indeed, batch-processing systems by their automation of parts of the operator’s work led to another configuration and operation of the computer room. This has traditionally been described as the transition from ‘open shop’ to ‘closed shop’. This terminology was borrowed from trade union contracts, in a ‘closed shop’ a worker had to join a union and there were restrictions to the kind of job he was allowed to do.¹² In the ‘open shop’ configuration, one could bring the program to the computer, run the program (or let the operator run it), and after execution bring the results back to your desk in the form of a printout. In the ‘closed shop’, the program was brought to the operator who put the program in a batch, and you had to wait until your batch was executed to go back and get your results. The ‘closed shop’ configuration is thus closely tied up with a separation of tasks. The programmer prepares a

¹² I thank one of the anonymous reviewers for pointing this origin out.

program, the operator handles the computer. This organisation is locked into position with its automation through classic batch-processing systems.¹³

The year 1956 is often quoted as the birthyear of the ‘first’ operating system (or rather, batch-processing system), though the genealogy of the batch-processing systems starts a bit earlier, viz. with Owen Mock’s 701 Monitor [49] or Moncreiff’s IBM 702’s Supervisor [50]. The idea matures when the General Motors - North American Aviation Monitor (short: GM/NAA monitor) for the IBM 704 (1956) [56] is developed and shared through the SHARE community. Its core program, the so-called Mock-Donald monitor, would be recycled, upgraded and implanted into later, more ambitious operating systems such as the SHARE operating system for the IBM 709 (SOS 1959) or the RAND-SHARE Operating System for the IBM 7090 (1962). The working of the most primitive batch system for the IBM 701 is described by Owen Mock as follows:

Multiple jobs were placed on a single 727 tape that became a batch whose target duration was one hour. There was a small in core resident monitor and a single system library and control program tape that also acted as backup for the resident monitor. Output was stacked on an output tape that could be removed and replaced if necessary. Upon the completion of a batch, the input and remaining output tape were removed and replaced with the next batch, and the output tape was taken to the 717 to be printed. [49, p. 794]

For the GM/NAA monitor this was complexified by splitting this up in a three-phase process: “an input-translation phase which converted data from decimal to binary, and programs from source to object language; an execution phase which was almost exclusively under the programmer’s direct control; and an output translation phase which processed line printer output, punched card output (both decimal and binary), and accounting records.” [56, p. 802] With time, monitors would produce longer batches and take on more and more complex and diverse tasks besides the linking, loading and compiling, such as converting formats, controlling I/O, producing reports and error logs etc. This evolution towards ever more complex monitors would go further and end with IBSYS (1962-1965) that as “a monitor of monitors [...] includes several of the older systems.” [35, p. 24] Indeed, this is visually evident in the IBSYS-diagram where the monitor of the operating system oversees the other subsystems, including programming languages, I/O control, a routine library system and a file system (Figure 1).

¹³ It should be noted that another interpretation of ‘open shop’ versus ‘closed shop’ exists (though it remains compatible with its trade union origins). In that interpretation, the ‘closed shop’ is the situation where only the operators and the machine code programmers can use the machine because the other users don’t know how to write in machine code. The ‘open shop’ situation then is when other users, now using a programming system, can start writing programs. These programs may possibly be executed in batches, see e.g. [15] for such an ‘open shop’ system using FMS where the users are empowered by the FORTRAN programming language.

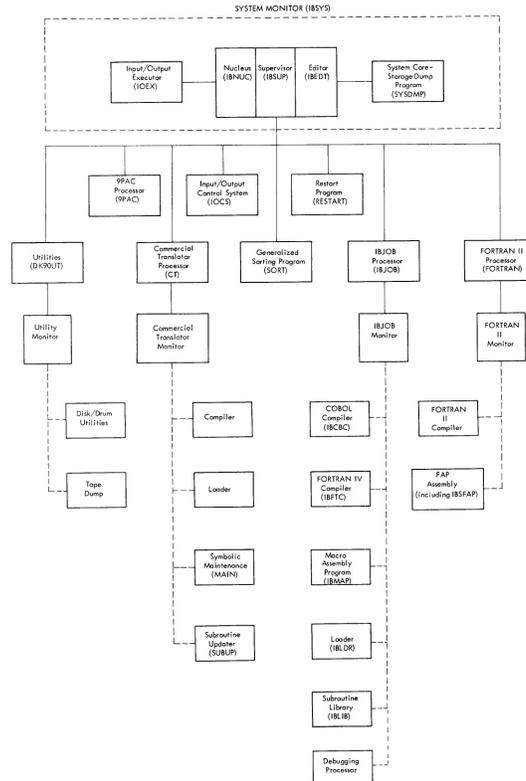


Fig. 1 A block diagram of the organisation of IBSYS (1962)

2.3 Integrated systems

In the same volume of the IRE Transactions that featured Moncreiff’s Supervisor, another paper, or rather abstract, talked of a ‘utility program system’ “to assist the coding, check-out, maintenance, and documentation of large-scale control programs.” [13, p. 21] This system was part of the ‘Comprehensive System’ that was developed from 1953 onwards for the Whirlwind computer at MIT. As one of the fastest and biggest computers of its time, Whirlwind had accumulated many programs over time and the Comprehensive System was a way to make the combined results of this team effort more generally accessible to the programmers. Since the groups of programming systems and utility programs on a computer mostly came from a variety of sources and had been adapted, recycled and reformatted a number of times, they were often an incoherent bunch. Finding ways to streamline the access to these programs and possibly making them work together was not trivial

at all. The idea of a programming system that eases access to various groups of programs, thus facilitating or partially automating the (human) programmer's work, presents another strand in the history of operating systems.

Another term popular in the late 1950s, referencing the same 'comprehensive' philosophy of the Whirlwind team, was 'integrated system'. It seems to have been used in particular by people associated with Ramo-Woolridge.

the programmer communicates information to the machine on the detailed level of his program data. In the integrated computation system this amount of information communicated is expanded to include items which otherwise would have to be communicated by word of mouth or by written instructions to the machine operator. The important concept here is that all items are integrated together to form one computation system to the exclusion of the use of the machine with isolated subsystems. [8, p. 8]

Or in the definition of the Handbook for Automation, Computation and Control "Interconnection of some or all these different utility programs into an organized, programmer-controlled, semiautomatic or automatic whole is usually called an integrated system." [33, p. 184]¹⁴ The examples quoted are MIT's CSSR (Comprehensive System of Service Routines) and MAGIC (Michigan Automatic General Integrated Computation). The same term was used by the Ramo-Woolridge team headed by W.F. Bauer that developed the so-called 'integrated computation system' for the ERA-1103 (1955). As W.F. Bauer wrote, this is "an over-all system to optimize the use of the computer in reducing programmer, computer, and clerical time in bringing problems to the production stage." [6, p. 181]

Integrated systems often included batch-processing facilities (as already on Whirlwind), but their emphasis was less on forming and sequencing batches (viz. on automating the operation of the computer), as on assisting in programming the computer. Rather the library of routines, or the so-called utility programs featured as the core of the system. The integrated system is mainly there to facilitate access to these, by providing input-output routines, conversion routines, sequencing routines etc. As the operating systems veteran George H. Mealy¹⁵ would later reflect, this is now also incorporated in the modern operating system:

Many functions now classed as OS functions were first embodied as utility subroutines and programs... Today, the library is an integral part of the OS – to the extent, for instance, that many programmers identify the UNIX system with its library rather than with its nucleus and shells. [48, p. 781]

Integrated systems are generally turned towards easing and shaping the user's interaction with the computer. Instead of the 'load, assemble, compile and

¹⁴ This quote comes from a section written by John Carr III.

¹⁵ After his involvement with Bell Labs' BESYS-systems and the SHARE community, he went to RAND where he headed the team that made the RAND-SHARE operating system. Afterwards, he worked for IBM on the OS/360 system.

execute' cycle typical of batch systems, integrated systems often relied more on interpretative systems.

Contrary to compiling programming languages, an interpretative system interprets each line of the stored program as it comes along. “The jump instructions in the main program which formerly directed control to the subroutines are eliminated”, and so the the control remains within the subroutines that “are all welded into one, an interpretive subroutine, which includes also a section to supervise the sequence in which the various operations are performed”, thus “ the instruction code of the machine is not merely augmented, it is entirely replaced.” [1, p. 16-3] Or, as the ACM Glossary (1954) had it, “An interpretive routine is essentially a closed subroutine which operates successively on an indefinitely long sequence of program parameters (the pseudo-instructions and operands).” [37, p. 18]

Many of the earliest programming schemes were interpretative, such as 701 Speedcode, Univac's Shortcode or MIT's Summer Session computer. Though they are costly in function of machine time, they can be used to use programming time more efficiently. This is especially the case when one is testing or debugging a program, but also in situations when subroutines have to be called very frequently. Many specialized interpretative routines were developed, e.g. for doing floating-point arithmetic or complex arithmetic, or for housekeeping functions. Some computer manufacturers also marketed systems that were a kind of general interpretative routines, e.g., the Bendix G-15 had its Intercom 1000 system to address its many microprogrammed routines, or NCR 304 had its STEP system that covered tape label handling automatically. For both systems, the user had the choice to either program the machine through the interpretative routine, or rather program it on the machine level, trading in ease of programming for speed of execution.

In at least one particular case, a very general interpretative routine was devised that could be used as a kind of interface between the user and the computer, levelling the access to the computer. Such schemes were developed for MIT's TX-0 and TX-2 computers, where they were coupled with rather advanced interactive possibilities such as a flexowriter and a display subsystem with lightpen.

Large high speed memories have arrived. [...] It is time to review our present techniques and philosophies in planning and programming computer applications. [...] This memo describes a utility system that is basic in form but unique in the sense that it assists the programmer in debugging and modifying his program at the console. This is accomplished by moving the utility program in memory alongside the program to be debugged and providing direct communication between the utility system and the programmer. [31, p. i]

This Direct Input Utility System (1958) developed for the TX-0 had a coherent inner logic that highlights how, through software, the computer's facilities are made 'at hand' to the user at a console (see Figure 2 for its global structure). It underwrote a philosophy of direct man-computer interaction that would later influence McCarthy's vision of time-shared programm-

ing and the PDP-line of computers (see Section 2.6) prefiguring the modern command-line access to an operating system.

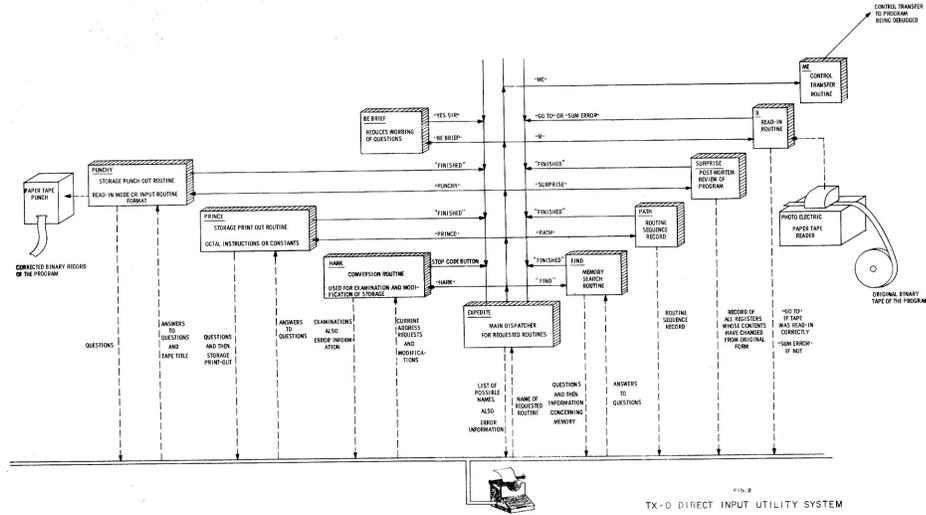


Fig. 2 A block diagram of the organisation of the TX-0 Direct Input Utility System (1959)

2.4 Special-purpose and real-time systems

2.4.1 Command and control systems

There were also, as W.F. Bauer later reminisced, “a number of special purpose systems, particularly command and control systems that utilized advanced operation system ideas ahead of their time” [10, p. 999]. The best-known (and most influential) of these systems is the SAGE system (Semi Automatic Ground Environment) that was developed during the 1950s. SAGE was a major project funded by the U.S. military to create a system of computers, networked through telephone lines, that had to coordinate radar data and information from defence sites to obtain a general picture of the airspace. This would help making decisions in the case of an atomic strike. MIT’s Lincoln Lab was part of the project as were its Whirlwind, TX-0 and TX-2 computers. IBM was also involved, building the massive AN-FSQ7 computers. A number of programmers from RAND founded one of the first software companies, SDC (System Development Corporation) to write the programs for

the project. The systems developed at MIT and IBM feature novelties such as real-time teleprocessing, and a display subsystem to accommodate interaction between the user and the computer using interrupts. The AN-FSQ7 computers also worked in duplex, a switch controlling the intercommunications between the two computers, and another switch regulating the communication with many peripherals.¹⁶ More generally, these systems could be called distributed systems because one (or more) central control units are coupled with a variety of peripherals with which they communicate in real-time. On these systems many ideas would be developed that could later be classified under multiprogramming, distributed computing or concurrent computing, but the most defining aspect of these systems at the time was the real-time character of operation.

Although SAGE and its related projects was probably the most influential cluster of special-purpose systems, there were many other special-purpose systems, both of military and of industrial nature. There were a number of digital-analog systems where one (or more) analog machine(s) was coupled with a digital processor. To organise the communication between the devices efficiently, special interfaces were developed to handle the synchronization by sequencing programs and signals through interrupts. The systems featured multiprogramming, intricate conversions and some complex scheduling routines.¹⁷ In the military field, advanced data processing units such as Ramo-Woolridge's 'polymorphic data system' RW-400 (1960) or Burroughs D-825 Modular Data Processing System (1962) were developed to control and direct a network of processors and devices. In the case of the D-825, a pioneering operating system, the Automatic Operating and Scheduling Program (AOSP), was developed too.

2.4.2 Process control systems

Another trend was the automation of industrial processes. Many special-purpose machines were developed in the 1950s and 1960s to control industrial processes, be it in a machine factory, an oil refinery, a power plant etc.¹⁸ It was widely understood that, in time, the controls and servosystems embedded in these machines might ultimately be replaced by (direct) digital control through the programming of a general-purpose digital computer. But most systems of the 1950s and 1960s were still of a mixed type, with analogue devices controlled digitally through so-called set points.¹⁹ In the 1950s,

¹⁶ There were other experiments in multi-computer systems around 1960, amongst them the Burroughs D-825 (1962) or the NBS's Pilot computer (1959).

¹⁷ See chapter 30 in [33] for some examples.

¹⁸ This technological evolution has to be contextualised socially, taking into account the tensions between organised labour, corporate management, technology and science, see [53].

¹⁹ An extensive state-of-the-art anno 1957 can be found in [32].

small general-purpose computers such as the Bendix G-15 or the LGP-30, were often used as the front-end for such data-processing in real-time. The later 1950s saw the development of a number of dedicated special-purpose systems that could be custom-programmed to control the processes and the machines of a factory or plant in real time. In this field, Thomson-Ramo-Woolridge with their RW-300 (1959), and later RW-330 (1961) computer offered a means for industrial control, together with expertise in programming executive routines. General Electric was also active with its GARDE system that used the GE-312 computer to control power plants (1959).²⁰ IBM entered the field rather late, in 1961, with their IBM 1700. In a whole different field, Bell Labs started to develop its Electronic Switching System (ESS) in the early 1960s, automating switching in the distributed telephone network through stored-program computing.

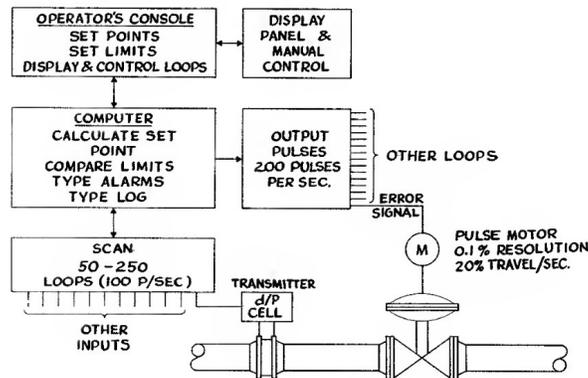


Fig. 3 Schematics of direct digital control through a process control computer.

2.4.3 Teleprocessing

Finally, the use of telephone lines as a means of communication and transmission between computers, marketed as ‘teleprocessing’ by IBM, also grew fast by the beginning of the 1960s. IBM’s first development along these lines were the IBM 057 and IBM 040 in 1941. The IBM 057 read cards and punched paper tape, then transferred the information telegraphically to the IBM 040 that punched cards again. The maximum rate of transmission was 3 cards per minute. Using its SAGE experience, IBM came up in 1960 with a much

²⁰ See also [22] for the transition from analogue to digital computing at Leeds & Northrup.

improved form of teleprocessing, using magnetic tape as carrier, with the IBM 7701 and 7702 Magnetic Tape Transmission Terminals processing 225 cards per minute. This was further improved upon with the IBM/360-line that was 100 times faster still [40, p. 5-6]. As with the dramatic acceleration of memory through magnetic tape in the 1950s, the acceleration of teleprocessing in the 1960s opened a world of new possibilities. IBM developed its real-time teleprocessing systems such as the well-known SABRE (1960, with American Airlines for air travel tickets reservations), or TOPS (1962, with Southern Pacific Railways). Other manufacturers such as Rand-Remington (with their Univac File Computer), Burroughs or General Electric developed their own brand of ‘teleprocessing’.

2.5 ‘Second-generation’ operating systems

Although the special-purpose programming systems are a rather hybrid bunch, the techniques developed within those systems, especially multiprogramming and real-time, but also forms of time-sharing, distributed computing and networking would prove to be valuable experience for later operating system design. Examples abound. The control system AOSP, originally developed for Burroughs’s military multi-computer-system D-825, would provide the blueprint for the Master Control Program (MCP) that Burroughs would use for its B5000-B5500-line of commercial computers. Some of Thomson-Ramo-Woolridge’s engineers who had worked in real-time process control or in the development of the military system RW-400, would later use their experience to design real-time systems at Honeywell and DEC. General Electric’s background in process control and the development of an operating system ECP (Executive Control Program) for the GE-312 would later prove valuable in the development of their time-sharing systems. In particular, they derived the Datanet-30 and its operating executive from their process control computer GE-312 to be used as a front-end processor to handle time-sharing on their GE-225 and GE-235 computers.

And, of course, in IBM’s OS/360 (1966) developed for its IBM/360 line of computers, the two threads of IBM’s development would merge. On the one hand the batch-processing system first developed by the SHARE community and later integrated into IBSYS, on the other hand IBM’s experience gathered during the SAGE project and commercialized in the SABRE and MERCURY systems.

as the use of computers extended into telecommunications, the function of the supervisory control program broadened to serve as the bridge between batch processing and service to remote locations. The resulting supervisory control program can be used to control a system which processes batch programs only, or a system dedicated to the control of telecommunications devices, or any combination of these two. [26]

For G.H. Mealy, the ‘first generation’ of batch-processing systems “oriented towards overlapped setup in a sequentially executed job batch” met up with dedicated machines of real-time applications leading up to “the basic structure of OS/360”. That structure is “equally applicable to batched-job and real-time applications, it may be viewed as one of the first instances of a ‘second-generation’ operating system. The new objective of such a system is to accommodate an environment of diverse applications and operating modes.” [47]

Though the term ‘second-generation’ systems was tailored to IBM’s particular evolution, one could, with a slight extension of meaning, also call operating systems such as Burroughs’s MCP or GE’s GECOS ‘second generation’ systems. They derive from older experiences in real-time processing, accommodate for batch-processing, and are flexible environments for a number of multiprogramming possibilities, time-sharing being one of them. In particular, these operating systems supported the modularity of the hardware design throughout, offering “hardware-software integration” as the B5500 publicity would call it. This was a two-way integration, the computers in their turn (IBM/360, GE-635/625 and B5000/5500) had to integrate interrupt and trap mechanisms, memory protection and memory management schemes, timers etc. to service the system’s needs.

2.6 Beginnings of time-sharing systems

Mature time-sharing systems fall just outside the chronological frame of this overview, but the conceptual development and the first experiments in time-sharing start around 1960. The very term ‘time-sharing’ was used frequently in the 1950s but always for denoting time-sharing in hardware. Mostly this meant that the main processor and peripheral equipment functioned simultaneously, sharing the computer’s time. The I/O buffer, I/O switches or the I/O channel are instances that enable such hardware time-sharing. One of the earliest proposals to let a processor not only control several terminal devices but also multiple program sequences quasi-simultaneously is described in a 1954-report by Wesley Clark [21, 30].²¹ The idea would be rediscovered and reformulated more than once during the late 1950s.

Bob Bemer, then working at IBM on developing programming standards, wrote in 1957 of a computer “servic[ing] a multitude of users” [11, p. 69]. Around the same time, Walter Bauer, project manager at Ramo-Woolridge, envisioned a conjectural computer, the Ultradatic [9, p. 49].

²¹ Clark worked at Lincoln Lab on the Whirlwind and the Memory Test Computer at the time they were starting to prepare for project SAGE, he would later head the development of the TX-0 and TX-2. Remark that human or interactive intervention is not planned in Clark’s 1954 proposal, though such interaction would appear on the TX-0 and TX-2.

each large metropolitan area would have one or more of these super computers [that] would handle a number of problems concurrently. Organizations would have input-output equipment installed on their own premises and would buy time on the computer much the same way that the average household buys power and water from utility companies.

Both Bemer and Bauer were heavily involved with early programming and operating systems, and their idea of what would be called the ‘computer utility’²² pursued the evolution they saw in computing systems. Computers and their peripherals had become ever faster at the risk of wasting computer time if not well-managed, and the operation and programming of computers had become increasingly automated. This could create a new kind of computer user. Inspired by the consoles and displays that had been developed in command and control systems such as SAGE during the late 1950s, they envisioned laymen-users (without operating or programming experience) requesting information from a central computer. Such systems could be expert systems, such as the mathematical Culler-Fried Online System [25] (1963) or the Hospital Research System at BBN (1965), or general information processing systems as the proposed Ultradatic or, later, project MAC and Multics.

A close though slightly different philosophy, focussing less on utility users but rather on programmers, was proposed by C. Strachey and J. McCarthy in 1959. The automatising of sequence control through hardware interrupts (leading to multiprogramming) had not only partially automatised the operator, but also kept the programmer away from the execution of his program. Their proposals sought to counteract this. Strachey lamented that the computer was used inefficiently, considering the high speeds of the processor, and of the input and the output. Therefore, he proposed “the concept of time sharing between operators [...] to allow [...] manual programme checking at a special console”, and also to do maintenance. To handle the complex coordination, Strachey described some hardware devices such as interlocks and interrupts, and a “Director [...] master programme designed to cater automatically for the conflicting demands of a number of stations of different types within a predetermined basic plan” [62]. John McCarthy, in part inspired by the interactive use of MIT’s TX-0 [45, p. 52], sought a system to accommodate his programming language LISP, that was planned to have interactive debugging facilities:

programmers are given the same near-simultaneous ability through time-shared computer use with routines designed to minimize programmer decision time. This involves connection of a number of typewriters to a computer, a language for communication, a program filing system which can allow fast access to one of many partially run programs in memory or on file, and a monitor program [and] could allow a ‘conversation’ between programmer and machine. [63, p. 12]

Instead of the rather ‘passive’ user of a computer utility, Strachey and McCarty saw the possibility of an ‘active’ user, communicating with the machine.

²² The ‘computer utility’ has recently (and anachronically) been reclaimed as a precursor to cloud computing.

The development in flexowriters, teletypes, displays and consoles made such interaction possible. As a consequence, new programming languages, dubbed ‘conversational languages’, would be created, either by adapting existing languages (IPL-V, MAD, APL, ALGOL) or by developing new ones (LISP, BASIC, Jovial).

Place	Computer	Year	System
MIT	IBM 709	Nov 1961	Experimental Time-Sharing System
BBN	PDP-1	Sep 1962	Time-Sharing System
MIT	PDP-1	May 1963	Time-Sharing System
MIT	IBM 7090	June 1963	Compatible Time-Sharing System
SDC	AN/FSQ32 with PDP-1	June 1963	Time-Sharing System
RAND	Johnniac	Jan 1964	Johnniac Open Shop System
Stanford	IBM 7090 with PDP-1	Jul 1964	Odin
DEC	PDP-6	Nov 1964	Time-sharing System
IBM	IBM 7094	Fall 1964	Time-Shared Monitor System
Dartmouth	GE-235 with Datanet 30	Fall 1964	Dartmouth Time-Sharing System
Carnegie Institute	G-20	Fall? 1964	Carnegie Institute Timesharing
Berkeley	SDS-930	April 1965	Berkeley Time-Sharing System

Table 2 An overview of the first time-sharing systems 1960-1965

A first experimental demonstration of time-sharing, building up from FMS and adding modified flexowriters, was performed at MIT on an IBM 709 in 1961 [23, 65]. Backed by J.C.R. Licklider’s enthusiasm for time-sharing, this demonstration would lead ARPA to sponsor a number of time-sharing projects across the U.S.A.²³ From that moment onwards, a number of experimental installations sprang up in the U.S. (see Table 2), many building upon experiences gathered during the SAGE project (MIT, DEC²⁴, SDC, IBM) or in multiprocessing and process control (GE, CDC²⁵, SDS²⁶). The first stable versions only appeared late 1963 and 1964 [59, pp. 90-91]. As for commercial versions of timesharing, DEC, GE, IBM, SDS and CDC only started offering commercial access to interactive time-sharing late 1965 [24]. This also marked the beginning of the discussions between time-sharing partisans and

²³ For a history of ARPA sponsored research in timesharing and its eventual influence, see [66, Chapter 5 & 6].

²⁴ DEC’s PDP-1 owed much to the design of the TX-0, also its engineers closely communicated with MIT and BBN for the development of their time-sharing systems. DECUS, the PDP user’s group, would play an important role in spreading the implementation of time-sharing on the PDP-machines.

²⁵ An important part of CDC’s personnel, in particular William Norris and Seymour Cray, came from ERA where they had worked on the ERA 1103 and the Naval Tactical Defence System (NTDS), a computerized information processing system.

²⁶ Max Palevsky and Robert Beck, who founded SDS in 1961, came from Bendix and Packard-Bell where they had been involved in developing computers such as the Bendix G-15 and the PB-250 that were often used as process control computers.

batch-processing advocates²⁷, or, to put it differently, from an economic perspective, it started off “a gargantuan contest to dominate the multibillion dollar software industry” [58, p. 8].²⁸

3 IBM invents the ‘operating system’

Although the term ‘operating system’ is now the prevailing term²⁹, other terms were in use. As Orchard-Hays remarked in his 1961-overview, various names were used to designate the ‘master’ routine of an operating system:

A number of terms have come into use for parts of an operating system. The term ‘supervisory program’ has already appeared above. The supervisor is the program which maintains ultimate control of the machine at all times and to which control reverts when a routine finishes its function or when an unexpected stop or ‘trap’ occurs. Terms which are used more or less synonymously with ‘supervisor’ are ‘executive routine,’ ‘monitor,’ ‘master control routine.’ [55, p. 290]

Many variants on these names exist, such as ‘control sequence routine’, ‘executive control’, etc. The names for these routines were often used in an extended way to designate the whole system.³⁰ People talked of ‘executive system’, ‘monitor system’, ‘supervisory system’, ‘control system’, ‘program sequencing system’, etc. instead of ‘operating system’.

How did ‘operating system’ become the term of preference? The very term itself seems to have been coined in the SHARE community and was first used to denote a specific system with the development of the SHARE Operating System (SOS) by a SHARE committee. In the issue of the *Communications of the ACM* devoted to the SHARE system the term is not used, instead the ‘SHARE 709 System’ is presented. As D.L Shell noted, “The initial problem facing the committee was to define what was meant by a system”, but it should be “generally acceptable to all of the users of this particular machine.” [60, p. 124 and p. 126] As to the controlling part of the system, the

²⁷ The 1965 issues of both trade magazines *Computers and Automation* and *Data-mation* amply illustrate the early discussions.

²⁸ This aspect of the time-sharing industry is closely connected to the turn towards viewing programs as a commodity and the emergence of the software industry in the 1960s, see [18], [34] and, for time-sharing in particular, [19].

²⁹ It should be remarked that in other languages (and thus countries), sometimes different terms have prevailed. In many languages, such as Spanish, Italian, Swedish or Russian, a variant of ‘operating system’ is used, but in Germany, ‘Betriebssystem’ is the usual word, in France, ‘système d’exploitation’, in the Netherlands ‘besturingssysteem’.

³⁰ This transfer of meaning, from a part of a system to the whole system, is quite a natural linguistic process called ‘pars pro toto’ (the parts for the whole) or ‘metonymy’. Some everyday examples of this process are: ‘I read the latest Stephen King’ (the author stands for the book), ‘Berlin expressed its support with the French people’ (Berlin, as a capital, standing for Germany or its government).

‘supervisory control program’, it “coordinates the use of the various parts of the SHARE 709 System and is responsible for maintaining the computer in continuous operation during the processing of a group of independent jobs.” [14, p. 152] It “provides a standard formulation of a job in respect to machine operation” and eliminates “wasted ‘between-job’ time.” This corresponds rather exactly to the advantages of batch-processing systems, though neither the term ‘batch’ nor ‘operating system’ are used.³¹

In the manual for the SHARE community, the SHARE 709 System is called SOS (for SHARE Operating System) throughout. In the introduction it says:

the SHARE operating system, familiarly known as SOS, is a highly flexible complex of languages, procedures and machine codes. The threefold purpose of the System is to provide assistance to the 709 programmer in the coding and check-out phase of program preparation, to assume from the 709 machine operator those burdens that may be sensibly automated and to provide the computer installation with an efficient operation and complete and accurate records on machine usage [36, sec. 01.01.01]

They remark that “SOS is in reality an integrated system, it has for convenience and easy reference been divided into the following subsystems”. These are: The SHARE-Compiler-Assembler-Translator (SCAT); The Debugging System; The Input/Output System; Monitor.

It seems that nomenclature in official ACM publication and in practice did not completely coincide for the IBM 709 System resp. SOS. However, it is clear from both sources that ‘operating system’ had not yet imposed itself as the normal term, and that its definition was still in the realm of ‘programming system’ or even ‘integrated system’. This changed with the systems that would follow the IBM 709 System (resp. SOS). It is already evident in the Fortran Monitor System (FMS), developed 1959 by North American Aviation. FMS was SOS’s main contender as an operating system within the SHARE community and would prove to more successful than SOS.³² As the FMS manual specifies: “the Monitor is a supervisory program for 709/7090 FORTRAN, FAP, and object programs. It calls in the various System programs as needed.” [38, p. 61] Whereas SOS was conceived as a kind of programming language, FMS was, from the beginning, conceived as a loader and linker for FORTRAN programs. This helped to clearly distinguish between the ‘operating system’ and the ‘programming system’.

³¹ As a matter of fact, in the ACM-publications on the SHARE 709 system, the term ‘operating program’ is used to denote the program running on the machine. This use of ‘operating’ makes the use of ‘operating system’ if not impossible, at the very least confusing.

³² In 1961, 76 % of IBM 709 and 7090 installations used FMS [44, p. 819]. One of the main reasons of SOS’s lack of succes was its failure to accommodate for FORTRAN usage, another one the complexity of its command language, cfr. [4, pp. 731-733].

Perhaps it was exactly the succes of FORTRAN and its profiling as a ‘programming language’³³ that made it possible to separate the programming system from the operating system. Even if this separation is somewhat artificial and problematic from a holistic point of view, the fact that there was a clearly recognizable ‘package’ that was the programming language FORTRAN (and its system comprising assembler, compiler and libraries), and that there was another package that eased the access to and use of FORTRAN in conjunction with the hardware components and other programming systems, surely added to the distinguishability of ‘operating system’.

This evolution in thinking is made explicit by George H. Mealy who was part of the programmer’s team at RAND to improve on SOS so it would accommodate FORTRAN, this resulted in the RAND-SHARE Operating system. In his report on ‘Operating Systems’, Mealy wrote:

The object of having a machine is to run jobs, not programming systems. To call the systems that stand between the programmer and the machine “programming systems” is to place undue emphasis on mechanical coding aids and not enough emphasis on the other aspects of operation. By “operating systems” we shall mean the whole complex of programming, debugging and operational aids with which the programmer deals. [46, p. 4]

This way, ‘operating system’ came to encompass and control more and more the programming system(s) of a computer. In quite the same way, Bob Bermer, at the time working for IBM, saw the operating system as Phase III in the development of programming systems, the operating system literally encompassing and controlling the programming system(s) [12].

A similar separation between programming and operating system is noticeable in the introduction the RAND-SHARE Operating system manual:

An operating system is a complex of computer routines which are used to get programs and data into and out of the machine, transform data (including program assembly and compilation), supervise job and task sequencing, and facilitate the communication between the programmer and components of the operating system [17, p. iii]

As for SOS, the purpose of the RAND-SHARE system is threefold: ‘Machine time savings; Operational efficiency; Programmer time savings.’ [17, p. 5] But now, the human operator is faded out in the description, and the operating system starts to govern the programming system(s). This leaves the operating system as the main interface between the programmer and the computer and lets the operator disappear (at least in theory, certainly not in practice!).

This trend deepens with the operating sytem IBM will develop, IBSYS.

³³ The idea of programming *language* seems to have first developed in the user’s communities, notably USE (1955), and later proliferated. The emphasis on ‘language’ probably helped to stress that it was a coding technique that was universal and portable, cf. [54]. If one looks at FORTRAN in particular, a distinction is made within the FORTRAN system between the language, in which programs are written, and the translator.

The 7090/7094 ibsys Operating System consists of an integrated set of system programs operating under the executive control and coordination of the System Monitor. The System Monitor, by coordinating the operation of the subsystems, allows a series of unrelated jobs to be processed with little or no operator intervention. By reducing the degree of human participation in the mechanics of data processing, the 7090/7094 ibsys Operating System ensures that jobs are processed faster, more efficiently, and with less likelihood of human error. As a result, turn-around time (i.e., the interval between the time a programmer submits a job for processing and the time he receives results) is significantly reduced. [39, p. 5]

The description literally removes the operator from the equation and posits that the operating system will act as a catalyst for the programmer's work. The word 'integrative' marks how the many program groups are put into a single configuration in which they can be easily accessed and used, but does not imply that they are on the same level. Rather, the operating system, and particularly its monitor, takes on the hierarchical top position in the configuration of the computer and its users. It controls how (program) data are processed and reduces problems related the human intervention. It also controls the other programming systems, making an 'operating system', to definition, "a group of programming systems operating under the control of a monitor program [28, p. 631]

The same hierarchy perspires from the block diagram of the IBSYS system (Figure 1). The IBM solution of the early 1960s spelled out most clearly how to install buffering layers around the machine and its systems making the operating system the main interface for the human user, easing access to the computer and its facilities. There was the IOCS (Input Output Control System) handling the I/O communication and buffering information, and on top of that there was the operating system IBSYS. IBSYS controlled programming systems such as FORTRAN and COBOL, subroutine libraries and I/O routines, and older batch systems such as FMS. There is a system for each of the tasks that could be fully automated, and the operating system supervising them all, replacing the human operator (at least in theory). New facilities, such as teleprocessing, were accommodated under the supervising monitor.

Though a 'second-generation' system supporting multiprogramming, also IBM's later system system OS/360 (1966) bears in its general outline the same philosophy. The user is kept at a distance from the machine via a number of layers, including the human operator and the operating system, while a supervisor handles all communication between the user's side and the groups of programs and programming systems (see Figure 4).

Many other operating systems in the mid 1960s did not adhere to this hierarchical structure, mostly because they featured multiprogramming and/or (real-time) interaction with the user prominently. E.g., Univac's EXEC I (1962) has a communication processor and a scheduling routine as its central components; CDC's SIPROS system (1965) has a pool of peripheral processors that either do I/O or sequencing under the guidance of a monitor;

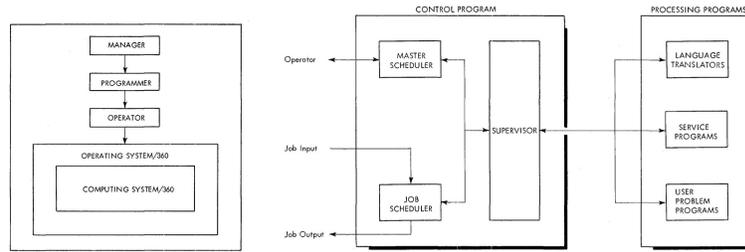


Figure 1. System/360

Figure 2. General Organization of Operating System/360

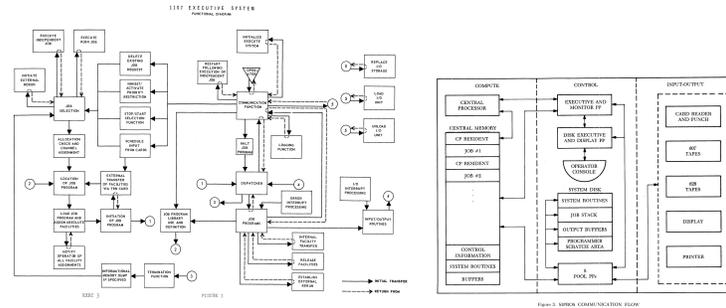


Fig. 4 Block diagram of the organisation of some mid 1960s operating systems. On top, two diagrams from OS/360 (1966); below a diagram from Univac’s EXEC (1962) and from CDC’s SIPROS (1965). IBM’s philosophy of ‘layering’ the user’s access to the machine perspires clearly from the drawing top left, and the hierarchical structure within OS/360 with the supervisor in the middle comes out well in the diagram top right. In contrast, the diagrams of EXEC and SIPROS display more complex relations between a variety of units.

Burroughs’ MCP (1965) has a scheduler at its heart that organizes and manipulates the tables that contains the essential parameters of the computer and programming systems (see Figure 4). Indeed, the structure of operating systems would become ever more complex in the years after 1964, but the name ‘operating system’ would stick, even if the new systems would do a lot more than merely automating the human operator.

4 Conclusion

If one does a literature search on book publications that have ‘operating system’ in their title, the first appearances are in the late 1960s and there is a clear peak during the 1970s. This is definitely a symptom of the importance of the topic after the emergence of time-sharing and of software engineering. Indeed, operating systems are one of the main examples of large programs that spurred the need for a more systematic approach to software,

an approach that came to be called, though not without animosity, software engineering. OS/360 and Multics both play an important role as paradigmatic cases of operating systems. How to handle real-time operation, concurrency of programs and multiple users on one computer are central problems of the operating systems of the late 1960s. Basic concepts and design techniques, such as segmentation, file systems, virtual memory, scheduling algorithms etc., were developed, as were general operating system philosophies, as e.g. virtual machines, hierarchical system or a kernel-based system. Vocabulary stabilized and people started writing about ‘operating systems’ proper.

Before the late 1960s vocabulary nor techniques had become part of a larger consensus, be it in the industry or in academia. Even the term ‘operating system’ itself had not yet imposed itself, though most computer manufacturers saw the necessity of including something like it in their ‘computer package’. Indeed, something like an operating system was badly needed to assist the human user so as not to slow down the automatic operation of the computer and to exploit fully the latest technological advances. The reciprocal development of hardware and software between 1954 and 1964 made both the computer system and the programming system more powerful and more versatile, but at the same time more complex and less surveyable. With the availability of larger and faster memory devices such as magnetic tapes and random-access disk drives (in combination with interrupts and I/O buffers) on the one hand, and with the parallel development of advanced programming systems on the other hand, the human element, be it a programmer or an operator, was completely outperformed by both the speed and amount of information processed by the computer.

With the trend towards automatic programming systems, also the first operating systems appeared. Though in the 1950s it is often difficult to distinguish clearly between programming systems, utility programs or control programs, the creation of batch-processing systems and the arrival of multiprogramming made the concept of an operating system slowly more distinct. Although the term ‘operating system’ itself stems from IBM and its users and is closely connected with a monitored batch-processing system (see Section 3), parallel developments equally brought about the distinguishability of the operating system. The integrative approach that brought groups of often unrelated programs into a more structured configuration helped to define the contours of an operating system. IBM even used the term ‘integrated’ for its IBSYS system to stress *a posteriori* how all groups of programs, old and new, were made to work together under the supervision of the monitor [52]. A final, important though often unnoticed factor in the emergence of operating systems were the special-purpose systems, notably the command and control systems and the process control systems. The experience gained with these systems in real-time operation and multiprogramming was important, in particular the development of special-purpose control programs to handle asynchronous communications. These communications could happen between the peripherals and the central processor, but also between human input on a

console or interactive device and a processor. The know-how acquired in this field would provide important cues for the later development of time-sharing systems.

On a more general level, the emergence of the first operating systems is a part of the rise of systems software and the software industry. The turn from user-driven software to manufacturer’s software is one aspect of this, the debate about the place of the user and the use of software in the batch-processing versus time-sharing discussions another. Also, the development of operating systems would take paradigmatic value in the later software crisis. But the early 1960s showed foremost that a concept of an operating system had become viable, and, as the developers of Burroughs’s AOSP had it, the operating system had become “as much a part of the [computer’s] system structure as is the actual hardware”. More even, according to them, there had been “a change of perspective”, viz. “computers do not run programs, [...] programs control computers.” [5, p. 95]

Acknowledgements I would like to thank Baptiste Mèlès for inviting me to talk about Multics in his seminar *Codes Sources* and I. Astic, F. Anceau and P. Mounier-Kuhn for giving me the opportunity to expand on operating systems before 1964 at the CNAM seminar on the history of computing. Doing some research for these talks and for my course *Introduction to the History of Computing* at Paris 8 was the start for this study of early operating systems. Finally, I would like to thank the organizers of the third HAPOP colloquium in Paris where this paper was first presented as well as Liesbeth De Mol for discussing the paper with me during the writing process. Finally, my thanks go to two anonymous reviewers whose comments helped to improve the paper.

References

1. Adams, C.W.; Gill, S. and others (eds.): Digital Computers: Business Applications. Summer program 1954,
2. Adams, C.W.: Developments in programming research. AIEE-IRE ’55 (Eastern) Papers and discussions presented at the the November 7-9, 1955, eastern joint AIEE-IRE computer conference, pp. 75-79 (1955).
3. Adams, C.W.: A batch-processing operating system for the Whirlwind I computer. AFIPS Conference Proceedings, vol. 56, pp. 785-789 (1987).
4. Akera, A.: Voluntarism and the fruits of collaboration: The IBM user group Share. Technology and Culture, 42(4), pp.710-736 (2001).
5. Anderson, J.P.; Hoffman, S.H.; Shiman, J and Williams, R.J: The D-825, a multiple-computer system for command & control, 1962 Fall Joint Computer Conference (AFIPS), pp. 86-96 (1962).
6. Bauer, W.F.: An Integrated Computation System for the ERA-1103. Communications of the ACM, 3 (3) pp. 181-185 (1956).
7. Bauer, W.F. and West, G.P.: A system for general-purpose digital-analog computation. Communications of the ACM, 4 (1) pp. 12-17 (1957).
8. Bauer, W.F., Use of Automatic Programming. Computers and Automation, 5 (11), pp. 6-11 (1956).

9. Bauer, W.F.: Computer Design from the Programmer's Viewpoint. Proceedings Eastern Joint Computer Conference, December 1958, pp. 46-51.
10. Bauer, W.F. and Rosenberg, A.M.: Software Historical perspectives and current trends. Fall Joint Computer Conference, 1972, pp. 993-1007 (1972).
11. Bemer, R.: What the Engineer should know about Programming: How to consider a computer. Data Control Section, Automatic Control Magazine, 1957 March, pp. 66-69 (1957).
12. Bemer, R.: The Present Status, Achievement and Trends of Programming for Commercial Data Processing. In: Digitale Informationswandler, ed. Hoffmann, Wiesbaden, pp.312-349 (1962).
13. Bennington, H.D. and Gaudette, C.H.: Lincoln Laboratory Utility Program System. AIEE-IRE '56 (Western) Papers presented at the February 7-9, 1956, joint ACM-AIEE-IRE western computer conference p.21 (1956).
14. Bratman, H and Boldt, I.V.: The SHARE 709 System: Supervisory Control. Communications of the ACM, 6 (2), pp. 152-155 (1959).
15. Breheim, D.J.: 'Open Shop' Programming at Rocketdyne Speeds Research and Production. Computers and Automation, 10 (7), pp. 8-9 (1961).
16. Brinch Hansen, P.: Classic Operating Systems: From Batch Processing to Distributed Systems. Springer, Berlin (2001).
17. Bryan, G.E.: The RAND Share operating system manual for the IBM 7090. Memorandum RM-3327-PR, Santa Monica, CA (1962).
18. Campbell-Kelly, M.: From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry. MIT Press: Cambridge, MA. (2003)
19. Campbell-Kelly, M. and Garcia-Swartz, D.D.: Economic Perspectives on the History of the Computer Time-Sharing Industry, 1965-1985, IEEE Annals of the History of Computing, 30 (1), pp. 16-36 (2008).
20. Ceruzzi, P.: A history of modern computing. 2nd edition. MIT Press, Cambridge, Massachusetts (2003).
21. Clark, W.: The multi-sequence program concept. Lincoln Lab Memorandum 6M-3144, 1954.
22. Cohn, J.: Transitions from Analog to Digital Computing in Electric Power Systems. IEEE Annals of the History of Computing, 37 (3), pp. 32-43 (2015).
23. Corbató, F. J.; Dagget, M. M.; and Daley, R.C.: An experimental time-sharing system. Proceedings AFIPS 1962 SJCC, ol. 21, pp. 335-344.
24. Computer Research Corporation: Time-Sharing System Scorecard. No. 1 Spring 1964, No. 2 Fall 1965.
25. Culler, G.J. and Fried, B.D.: An Online Computing Center for Scientific Problems. M19-3U3, TRW report (1963).
26. Dines R.S.: Telecommunications and supervisory control programs. Computers and Automation 15 (5), pp. 22-24. (1966)
27. Drummond, R.E.: BESYS revisited. AFIPS Conference Proceedings, vol. 56, pp. 805-814 (1987).
28. Fisher, F.P. and Swindle G.F.: Computer programming systems. Holt, Rinehart and Winston: New York (1964).
29. Frank, W.L.: Organization of a Program Library for a Digital Computer Center. Computers and Automation, 5 (3), pp. 6-8 (1956).
30. Fredkin, E.: The time-sharing of computers. Computers and Automation, 12 (11), pp. 12-20 (1963).
31. Gilmore, J.T. Jr.: TX-0 Direct Input Utility System. Memorandum 6M-5097-1 Lincoln Lab.
32. Grabbe, E.N. (ed.): Automation in Business and Industry. London: Wiley (1957).
33. Grabbe, E.N. and Ramo, S. and Woolridge D.E.: Handbook of Automation, Computation and Control, Volume 2. Wiley, New York (1959).
34. Haigh, T. Software in the 1960s as concept, service, and product, IEEE Annals of the History of Computing 24 (1), pp. 5-13 (2002).

35. Hassitt, A.: Programming and Computer systems. Academic Press: New York and London (1967).
36. Homan, C.E. and Swindle, G.F.: Programmer's Manual for the SHARE Operating system. IBM (1959).
37. Hopper, G.: ACM Glossary. (1954)
38. Reference Guide to the 709/7090 FORTRAN Programming System. IBM: Poughkeepsie (1961). (includes material from IBM 709/7090 FORTRAN Monitor, form C28-6065)
39. IBM 7090/7094 IBSYS system operator's guide. IBM: Poughkeepsie (1964).
40. IBM Field Engineering Education Student Self-Study Course: Introduction to Teleprocessing. IBM: Poughkeepsie (1965).
41. Knuth, D.E. and Pardo, L.: The early development of programming languages. Belzer, J; Holzman, A.G.; Kent, A. (eds). Encyclopedia of Computer Science and Technology. Marcel Dekker: New York, pp. 419-496. (1979)
42. Krakowiak, S. : Les débuts d'une approche scientifique des systèmes d'exploitation, Interstices, February 2014
43. Krakowiak S. and Mossière J.: La naissance des systèmes d'exploitation. Interstices, April 2013.
44. Larner, R.A.: FMS: The IBM FORTRAN Monitor System. AFIPS Conference Proceedings, vol. 56, pp. 815-820 (1987).
45. McCarthy, J.; Boilen S.; Fredkin E.; Licklider J. C. R.: A time-sharing debugging system for a small computer. Proc. AFIPS 1963 SJCC, vol. 23, pp. 51-57 (1963).
46. Mealy, G.H.: Operating Systems. RAND Report P-2584 (1962). Partially reprinted in [57].
47. Clark, W.A.; Mealy, G.H. and Witt, B.I: The functional structure of OS/360. IBM Systems Journal, 5 (1), pp. 3-51 (1966).
48. Mealy, G.H.: Some threads in the development of early operating systems. AFIPS Conference Proceedings, vol. 56, pp. 779-784 (1987).
49. Mock, O.R.: The North American 701 Monitor. AFIPS Conference Proceedings, vol. 56, pp. 791-795 (1987).
50. Moncreiff, B.: An automatic supervisor for the IBM 702. AIEE-IRE '56 (Western) Papers presented at the February 7-9, 1956, joint ACM-AIEE-IRE western computer conference , pp. 21-25 (1956).
51. Nelson, E.: Computer Installation at TRW systems – Some Experiences and Lessons. Computers and Automation 18 (8), pp. 21-22 (1969).
52. Noble, A.S. and Talmadge, R.B.: Design of an Integrated Programming and Operating System, I & II, IBM System Journal, vol. 2, pp. 152-181 (1963)
53. Noble, D.F.: Forces of Production: A Social History of Industrial Automation. New York: Knopf (1984).
54. Nofre, D; Priestley, M. and Alberts, G.: When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950-1960. Technology and Culture, 55 (1), pp. 40-75. (2014)
55. Orchard-Hays, W.: The Evolution of Programming Systems. Proceedings of the IRE, 49 (1), pp. 283-295 (1961).
56. Patrick, R.L.: General Motors/North American Monitor for the IBM 704 computer. AFIPS Conference Proceedings, vol. 56, pp. 796-803 (1987).
57. Rosen, S.: Programming Systems and Languages. New York: McGraw-Hill (1967).
58. Sackman, H.: Man-Computer Problem solving. Princeton etc.: Auerbach (1970).
59. Schwartz, J.: Interactive systems: promises, present and future, Proceeding AFIPS '68 fall joint computer conference, part I, pp. 89-98 (1968).
60. Shell, D.L.: SHARE 709 system: a cooperative effort. Communications of the ACM, 6 (2), pp. 123-127 (1959).
61. SHARE Operating System Manual, Distribution 1 to 5. Poughkeepsie: IBM (1960).

62. Strachey, C.: Time-sharing in large fast computers. Proc. International Conference on Information Processing, UNESCO (June 1959), Paris, paper B.2.19, pp. 336-341.
63. Teager, H. and McCarthy, J.: Time-shared program testing. Preprints of Papers ACM 14th National Meeting (Sept. 1959), 12-1 to 12-2.
64. Tanenbaum, A.: Modern operating systems. 2nd edition. Upper Saddle River, NJ: Prentice Hall (2001).
65. Walden D. and Van Vleck, T.: Compatible Time-Sharing System (1961-1973). Fiftieth Anniversary Commemorative Overview. IEEE Computer Society (2011).
66. Waldrop, M: The Dream Machine: J.C.R. Licklider and the Revolution That Made Computing Personal (2002).