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

Maarten Bullynck

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 defendors of batch-processing in the late 1960s. Important as these discussions were for thinking about the use and about the users of the computer, 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 systems[1] 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., [17, pp. 96-101], [46, pp. 6-18] or [33, 32]. Though also [14] 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[2] and identifies a set of questions and problems that drive the earliest history of operating systems. While adding more layering to the classic storyline, it also brings out the parallel developments, thus focussing in on the very question, what is an operating system?

1 Preconditions for operating systems 1954-1964

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, this changed for a number of reasons. New memory technologies became viable, both working and storage memories. The cheap magnetic drums were a good option to expand the capacity of working memory directly addressable by the computing unit, but 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 is essential for the development of operating systems.[3] While 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 [23, 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 or sequential (magnetic tape), or cyclic (magnetic drum).

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

A further refinement 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." [8]

This expansion of rapid storage for programmed routines goes hand in hand with the development of software. The latter half of the 1950s has traditionally been seen as the years software development took off [17, 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 (1957) that grew out of RAND's involvement with the SAGE project, or Computer Sciences Corporation (1959) etc. [16, 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. Other important systems, this time for 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. Second, from 1955 to 1957 an IBM team had been working on a scientific programming language that would eventually be called FORTRAN. As the

[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.

first fully developed programming language[5] 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 systems. They wanted to expand their functionality to include and accomodate 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 the overall system.

1.2 Changes in the mid 1960s

The years between 1962 and 1964 mark an endpoint to this first phase in the development of operating systems. The 'big' operating system projects OS/360 and Multics and especially the emergence of time-shared operating systems stand for this turning point, though they are rather the most conspicious 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 faster memory devices. 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. It did make scheduling of programs a necessity. The hardware interrupt made the first instances of multiprogramming possible and the introduction of I/O buffer memory made it proliferate in many directions. In a way, the idea of time-sharing a computer, viz. many users executing programs and using resources at the same time[6], can be considered as an extreme form of multiprogramming. However, 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. 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.

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 de-

[5] For the languages preceding FORTRAN, see [31].

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

veloped one.[7] Whereas 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 developed 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. Looking at some of the major computer manufacturers, they all came out with an operating system between 1962 and 1965 (see Table 1[8]). Some of these systems are rather primitive (GE's BRIDGE), others are rather classic batch systems (Philco's BKS[9] 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, e.g. CTSS on an IBM 709 and later on a PDP-1 (MIT), TS on a PDP-1 (BBN), JOSS on the Johnniac (RAND), the Cambridge multiple-access system on the Ferranti Atlas (Cambridge) etc. 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.

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)

[7] It is also in the early 1960s that the first overview articles on operating systems appear: [41, pp. 290-294] and [35].

[8] We did not include information on non-U.S. computers, 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] This system was actually developed by a user, the Bettis-Knoll power plant. It exploited the rather large I/O memory buffer and the executive control.

users (GM, NAA, Bell Labs, Michigan University...). 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. IBM also slowly integrated NAA's FORTAN Monitor System into their 709/7090 FORTRAN programming system (1960-1962) [34, p. 818-819] and then went on to produce their own operating systems, first IBSYS (from 1962 onwards), later OS/360 (1965 onwards). In parallel, user-driven developments of systems slowly waned.

2 What is an operating system?

2.1 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'.

The automation of parts of the operator's work also led to another configuration and operation of the computer room. This has traditionally been described as the transition from 'open shop' to 'closed shop'. In the 'open shop' configuration, one could bring the program to the computer, 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 the classic batch-processing system.[10]

[10] It should be noted that another interpretation of 'open shop' versus 'closed shop' exists. In that interpretation, the 'closed shop' is the situation where only the opera-

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 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 facilty 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." [39, 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." [39] 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.

1956 is often quoted as the birthyear of the 'first' operating system (or rather, batch-processing system), though genealogy of the batch-processing systems starts a bit earlier, viz. with Owen Mock's 701 Monitor [38] or Moncreiff's IBM 702's Supervisor [39]. The idea first matures when the General Motors - North American Aviation Monitor (short: GM/NAA monitor) for the IBM 704 (1956) [42] 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

---

tors 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. [13] for such an 'open shop' system using FMS.

> and remaining output tape were removed and replaced with the next batch, and the output tape was taken to the 717 to be printed. [38, 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." [42, p. 802] 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." [25, p. 24]

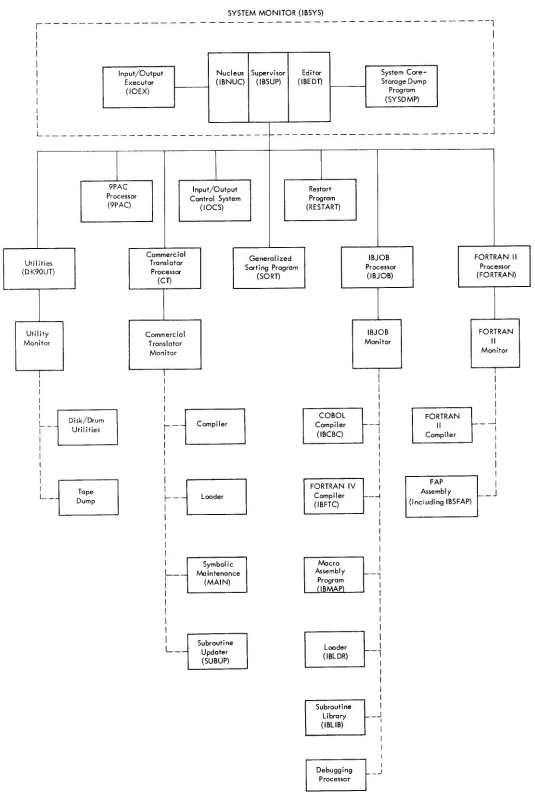

Fig. 1 A block diagram of the organisation of IBSYS (1962)

2.2 Integrated systems

In the same volume of the IRE Transactions that featured Moncreiff's Supervisor, another paper, or rather abstract, stood for another vision on what would become 'operating systems'. It talked of a 'utility program system' "to assist the coding, check-out, maintenance, and documentation of large-scale control programs." [11, p. 21] This system was part of the 'Comprehensive System' that was developed from 1953 onwards for the Whirlwind computer at MIT. The idea of a programming system that eases access to various groups of programs, thus facilitating or partially automating the (human) programmer's work, is another strand in the history of operating systems.

Another term popular in the late 1950s was 'integrated system'. It references the same 'comprehensive' philosophy of the Whirlwind team. 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. [7, 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."[24, p. 184][11] 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." [5, p. 181]

In the philosophy of 'integrated system' the program controlling the sequence of operation features less prominently than in the 'operating system' concept. Rather the library of routines, or the so-called utility programs feature as the core of the system. The integrated system is mainly there to facilitate access to these, by providing input-output routines, conversion routines and sequencing routines etc. As the operating systems veteran George H. Mealy[12] would later reflect, this is now also incorporated in the modern operating system:

---

[11] This quote comes from a section written by John Carr III.

[12] 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.

> 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. [37, p. 781]

In systems of the 'integrated' approach, more attention is generally devoted to ease and shape the user's interaction with the computer. Instead of the 'load, assemble, compile and execute' cycle typical of batch systems, 'integrated systems' often relied more on interpretative systems. In combination with enough fast memory and with a typewriter or flexowriter, this use of interpretative programming could sometimes have the feel of today's shell programming interface. In the special cases of the TX-0 and TX-2, where manual interaction with a display (cathode ray tube) was possible, computer usage via these interpretative systems prefigured some aspects of the interactive systems that would become viable with the time-sharing systems of the late 1960s.

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)." [27, 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. With the development of programming systems, comprising a macro-assembler or a fully developed language and compiling routines, these interpretative routines slowly disappeared. However, general interpretative routines that could be used as a kind of interface between the user and the computer were developed for some computers, provided enough (fast) memory to make them workable. 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. Especially the Direct Input Utility System for the TX-0 (see Figure 2 for its global structure) underwrote a philosophy of man-computer interaction that would later influence the PDP-line of computers. Also some computer manufacturers marketed systems that were a kind of general interpretative routines, e.g., the Bendix G-15 had its Inter-

com 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.

Problematically enough, this vision of 'comprehensive' or 'integrated' system' is obscured in historiography by its proximity to programming systems. Indeed, before 1962-64 (and even today), it is often hard 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) [22, p. 6]. While the 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'.[13] However, from another perspective, the operating routines oversee and control the programming systems, therefore it hierarchically is 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 is visually evident in the IBSYS-diagram where the monitor oversees the other routines (Figure 1), but is absent in a system of the 'integrative' kind such as TX-0's Direct Input Utility System (Figure 2).

2.3 Special-purpose systems

Although most early systems can be catalogized under either the 'operating' or the 'integrative' label, there were, 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" [9, p. 999]. The best-known (and most influential) of these systems is the SAGE system (Semi Automatic Ground Environment). SAGE was a major project funded by the U.S. military to develop 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 atomatic strike. MIT's Lincoln Lab was part of the project as were its Whirlwind, TX-0 and TX-2 computers. IBM

[13] See, e.g., the classic book by Rosen [43], but also [41], [21] or [25].

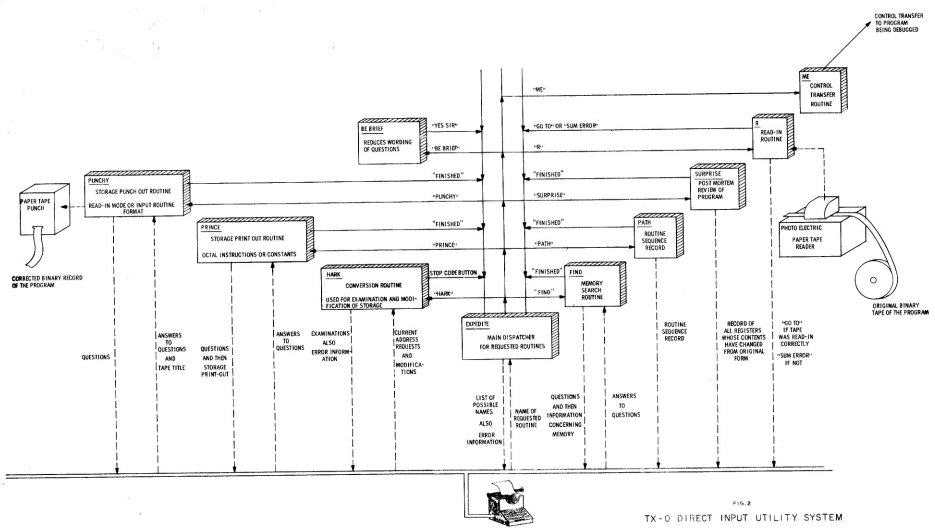

Fig. 2 A block diagram of the organisation of the TX-0 Direct Input Utility System (1959)

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 accomodate interaction between the user and the computer using interrupts. 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 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.[14] In the military field, advanced data processing units such as Ramo-Woolridge's

[14] See chapter 30 in [24] for some examples.

RW-400 (1960) or Burroughs D-825 (1962) were developed to control and direct a network of processors and devices. In the case of the D-825, a pioneering operating system, AOSP, was developed too.

Another, more general and eventually more lasting 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. 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.[15] Therefore, special-purpose systems had to be programmed on a central processor 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).[16] 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.

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, than 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 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 [30, p. 5-6]. As with the dramatic acceleration of memory through magnetic tape in the 1950s, the accelaration 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'.

Although the programming systems discussed above are a rather hybrid bunch, the techniques developed within those systems, especially multipro-

[15] An extensive state-of-the-art anno 1957 can be found in [23].

[16] See also [18] for the transition from analogue to digital computing at Leeds & Northrup.

gramming and real-time, but also forms of time-sharing, distributed computing and networking would prove to be valuable experience for later operating system design. Burroughs' AOSP system for the D-825 provided the blueprint for its later Master Control Program (MCP); Ramo-Woolridge's experience in real-time process control would later influence real-time system design at Honeywell and DEC; and General Electric's background in process control would laterprove valuable in the development of their time-sharing systems. And, of course, in IBM's OS/360 (1966) the two lines 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.[17]

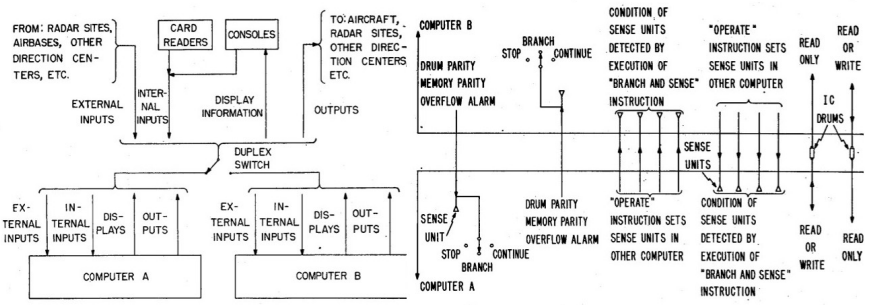

Fig. 1—Switching facilities.

Fig. 2—Intercommunication facilities.

Fig. 3 Schematics of switches that handle the communications between peripherals and the duplex SAGE computer (left); and an overview of intercommunication facilities between the two computers (right)

[17] G.H. Mealy would write that, compared to batch-processing systems (which he called first-generation), IBM's OS/360 was second generation, because it combined the ideas of batch-processing with real-time: "Then, as now, the operating system aimed at non-stop operation over a span of many jobs and provided a computer-accessible library of utility programs. A number of operating systems came into use during the last half of the decade. In that all were oriented toward overlapped setup in a sequentially executed job batch, they may be termed 'first generation' operating systems. A significant characteristic of batched-job operation has been that each job has, more or less, the entire machine to itself, save for the part of the system permanently resident in main storage. During the above-mentioned period of time, a number of large systems-typified by SAGE, MERCURY, and SABRE-were developed along other lines; these required total dedication of machine resources to the requirements of one 'real-time' application. By and large, however, these real-time systems bore little resemblance to the first generation of operating systems, either from the point of view of intended application or system structure. Because the basic structure of OS/360 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." [36]

3 IBM invents the 'operating system'

Although the term 'operating system' is now the prevailing term[18], 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.' [41, 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.[19] 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." [44, p. 124 and p. 126] As to the controlling part of the system, the '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." [12, 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.[20]

---

[18] 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'.

[19] 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 exemples 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 gouvernment).

[20] 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

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 [26, 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 succesful than SOS.[21] 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." [28, 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'.

Perhaps it was exactly the succes of FORTRAN and its profiling as a 'programming language'[22] 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

---

of 'operating' makes the use of 'operating system' if not impossible, at the very least confusing.

[21] In 1961, 76 % of IBM 709 and 7090 installations used FMS [34, 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].

[22] 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. [40]. 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.

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 distinguishibility 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. [35, 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 Bemer, 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) [10].

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 [15, p. iii]

As for SOS, the purpose of the RAND-SHARE system is threefold: 'Machine time savings; Operational efficiency; Programmer time savings.' [15, p. 5] But now, the human operator is faded out in the description, and the operating system starts to gouvern 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.

> 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. [29, 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 operating system, and particularly its monitor, also takes on the hierarchical top position in the configuration of the computer and its users. It controls and reduces human-related problems between (program) data and its processing. 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 [21, p. 631] The same hierarchy perspires from the block diagram of the IBSYS system (figure 1).

4 Discussion

If one does a literature search on book publications that have 'operating system' in their title, the first appearance 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. 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.

The IBM solution of the early 1960s spelled it out most clearly, it installed 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 accomodated under the supervising monitor.

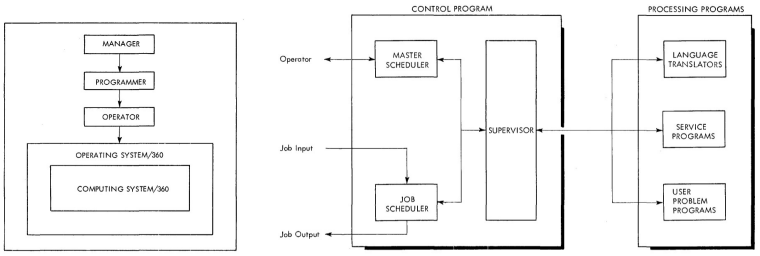

Figure 1. System/360

Figure 2. General Organization of Operating System/360

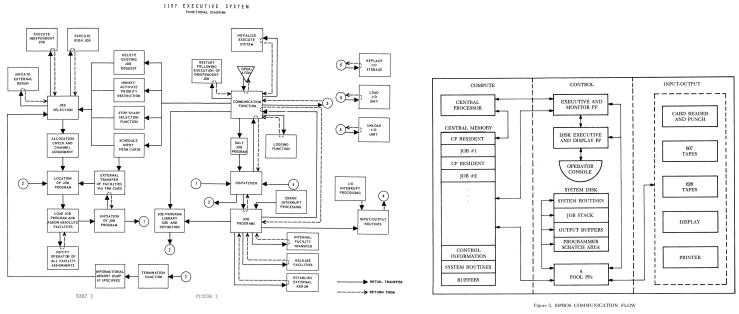

Fig. 4 Block diagram of the organisation of some mid 1960s operating systems. On top, two diagrams from OS/360 (1966); below a diagram from Univac's EXEC (1962) and from CDC's SIPROS (1965). IBM's philosophy of 'layering' the user's access to the machine perspires clearly from the drawing top left, and 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.

Many other operating systems around 1962 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; Burroughs' MCP (1965) has a scheduler at its heart that organizes and ma-

nipulates the tables that contains the essential parameters of the computer and programming systems. 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 did more and other things than a human operator would have done.

Although the term 'integrated system' did not catch on in the 1960s[23], the ideas behind it would remain alive in other forms. One form is the online computer system such as the Culler-Fried Online System of routines [19], another is the computer utility (such as the conjectural Ultradatic, see [8]). The first is a forerunner of modern expert systems, the latter has recently been reclaimed as a precursor to cloud computing. More importantly, because interactive facilities become more common in the late 1960s, not only through time-sharing, but also through the increased use of flexowriters and cathode ray tubes, the interpretative systems as pioneered on the TX-0 would foreshadow the user interface of such operating system as the Dartmouth Time Sharing System (1964), Multics (1969) or Unix (1972).

Acknowledgements I would like to thank Baptiste Mélès 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.

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. Bauer, W.F.: An Integrated Computation System for the ERA-1103. ACM, 3 (3) pp. 181-185 (1956).
6. Bauer, W.F. and West, G.P.: A system for general-purpose digital-analog computation. ACM, 4 (1) pp. 12-17 (1957).
7. Bauer, W.F., Use of Automatic Programming. Computers and Automation, 5 (11), pp. 6-11 (1956).

[23] Interestingly, the idea behind the 'integrated system' would be picked up by the IBSYS development team in 1963, although IBM would rather support the 'operating system' approach.

8. Bauer, W.F.: Computer Design from the Programmer's Viewpoint. Proceedings Eastern Joint Computer Conference, December 1958, pp. 46-51.
9. Bauer, W.F. and Rosenberg, A.M.: SoftwareHistorical perspectives and current trends. Fall Joint Computer Conference, 1972, pp. 993-1007 (1972).
10. Bemer, R.: The Present Status, Achievement and Trends of Programming for Commercial Data Processing. In: Digitale Informationswandler, ed. Hoffmann, Wiesbaden, pp.312-349 (1962).
11. 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).
12. Bratman, H and Boldt, I.V.: The SHARE 709 System: Supervisory Control. Communications of the ACM, XX, pp. 152-155 (1959).
13. Breheim, D.J.: 'Open Shop' Programming at Rocketdyne Speeds Research and Production. Computers and Automation, 10 (7), pp. 8-9 (1961).
14. Brinch Hansen, P.: Classic Operating Systems: From Batch Processing to Distributed Systems. Springer, Berlin (2001).
15. Bryan, G.E.: The RAND Share operating system manual for the IBM 7090. Memorandum RM–3327-PR, Santa Monica, CA (1962).
16. Campbell-Kelly, M.: From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry. MIT Press: Cambridge, MA. (2003)
17. Ceruzzi, P.: A history of modern computing. 2nd edition. MIT Press, Cambridge, Massachussets (2003).
18. 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).
19. Culler, G.J. and Fried, B.D.: An Online Computing Center for Scientific Problems. M19-3U3, TRW report (1963).
20. Drummond, R.E.: BESYS revisited. AFIPS Conference Proceedings, vol. 56, pp. 805-814 (1987).
21. Fisher, F.P. and Swindle G.F.: Computer programming systems. Holt, Rinehart and Winston: New York (1964).
22. Frank, W.L.: Organization of a Program Library for a Digital Computer Center. Computers and Automation, 5 (3), pp. 6-8 (1956).
23. Grabbe, E.N. (ed.): Automation in Business and Industry. London: Wiley (1957).
24. Grabbe, E.N. and Ramo, S. and Woolridge D.E.: Handbook of Automation, Computation and Control, Volume 2. Wiley, New York (1959).
25. Hassitt, A.: Programming and Computer systems. Academic Press: New York and London (1967).
26. Homan, C.E. and Swindle, G.F.: Programmer's Manual for the SHARE Operating system. IBM (1959).
27. Hopper, G.: ACM Glossary. (1954)
28. Reference Guide to the 709/7090 FORTRAN Programming System. IBM: Poughkeepsie (1961). (includes material from IBM 709/7090 FORTRAN Monitor, form C28-6065)
29. IBM 7090/7094 IBSYS system operator's guide. IBM: Poughkeepsie (1964).
30. IBM Field Engineering Education Student Self-Study Course: Introduction to Teleprocessing. IBM: Poughkeepsie (1965).
31. 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, pp419-496. (1979)
32. Krakowiak, S. : Les débuts d'une approche scientifique des systèmes d'exploitation, Interstices, February 2014
33. Krakowiak S. and Mossière J.: La naissance des systèmes d'exploitation. Interstices, April 2013.
34. Larner, R.A.: FMS: The IBM FORTRAN Monitor System. AFIPS Conference Proceedings, vol. 56, pp. 815-820 (1987).

35. Mealy, G.H.: Operatings Systems. RAND Report P-2584 (1962). Partially reprinted in [43].
36. 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).
37. Mealy, G.H.: Some threads in the development of early operating systems. AFIPS Conference Proceedings, vol. 56, pp. 779-784 (1987).
38. Mock, O.R.: The North American 701 Monitor. AFIPS Conference Proceedings, vol. 56, pp. 791-795 (1987).
39. 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).
40. 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)
41. Orchard-Hays, W.: The Evolution of Programming Systems. Proceedings of the IRE, 49 (1), pp. 283-295 (1961).
42. Patrick, R.L.: General Motors/North American Monitor for the IBM 704 computer. AFIPS Conference Proceedings, vol. 56, pp. 796-803 (1987).
43. Rosen, S.: Programming Systems and Languages. New York: McGraw-Hill (1967).
44. Shell, D.L.: SHARE 709 system: a cooperative effort. Journal of the ACM, 6 (2), pp. 123-127 (1959).
45. SHARE Operating System Manual, Distribution 1 to 5. Poughkeepsie: IBM (1960).
46. Tanenbaum, A.: Modern operating systems. 2nd edition. Prentice Hall, Upper Saddle River, NJ (2001).