Interrupt

Simon Yuill

2008

Originally published in Matthew Fuller (ed.), *Software Studies: A Lexicon*,
Cambridge, Massachusetts, London, England: The MIT Press, 2008.

In the early days of modern computing, the computer would execute a single program at a time, from start to finish. This is known as *batch processing*; programs would be collected in a batch and then run one after another. By the late 1950s a new paradigm had emerged, that of interactive computing, in which the computer operator could stop and start programs and edit them on the computer itself. This required the computer processor to receive external signals while it was running. Two methods emerged for handling this: *polling* and *interrupts*. In polling, the computer periodically checks to see if any external signals have arrived but the processor retains control over when they are handled. In interrupts, the signals are handled whenever they arrive, 'interrupting' the processor in whatever it is doing, and giving some control over its activities to an external agent. While polling continues to be used on some simple processor devices, the interrupt enabled more sophisticated forms of interaction between a computer and the external world. It has become the basis of most operating system designs and is hardwired into many processor chips and computer boards, such as the IRQ (Interrupt ReQuest) lines, which provide the link between the central processing unit (CPU) and all kinds of external devices such as keyboards, mice, and network cards. Interrupts can also be used for handling interaction between different programs on one operating system, signalling, for example, when a program has completed. It is also used for handling errors that arise in the execution of a program, such as buffer overflows, errors in allocating memory, or attempting to divide a number by zero. The interrupt is the main mechanism through which an operating system seeks to maintain a coherent environment for programs to run within, co-ordinating everything external to the central processor, whether that be events in the outside world, such as a user typing on a keyboard or moving a mouse, or things outside the system's internal coherence, such as a buffer overflow or an operational error in a piece of software.[1]

The interrupt fundamentally changed the nature of computer operation, and therefore also the nature of the software that runs on it. The interrupt not only creates a break in the temporal step-by-step processing of an algorithm, but also creates an opening in its operational space. It breaks the solipsism of the computer as a Turing Machine, enabling the outside world to 'touch' and engage with an algorithm.[2] The interrupt acknowledges that software is not sufficient unto itself, but must include actions outside of its coded instructions. In a very basic sense, it makes software *social*, making its performance dependent upon associations with *others* — processes and performances elsewhere. These may be human users, other pieces of software, or numerous forms of phenomena traced by physical sensors such as weather monitors and security alarms. The interrupt connects the dataspace of software to the sensorium of the world.

[1] The specific forms and namings of interrupts can vary on different operating systems and hardware platforms, for a detailed account of interrupt handling in Linux see Daniel P. Bovet and Marco Cesati, *Understanding the Linux Kernel*. For a comparison of interrupt systems on different processors, including those used in embedded systems, see William Bolton, *Microprocessor Systems, and Myke Predko, Programming and Customizing PICmicro MCU Microcontrollers*. For information on the historical development of the interrupt and comparisons to polling, see Mark Smotherman, "Interrupts"; Norman Hardy, "History of Interrupts"; Randall Hyde, "Interrupts and Polled I/O," in *The Art of Assembly Language Programming*; David A. Rusling, "Interrupts and Interrupt Handling."

[2] Peter Wegner and Dana Goldin have argued that this introduces a different level of capability into computers that the Turing Machine does not allow, therefore changing the ways in which computations can be processed and assessed; see Peter Wegner and Dana Goldin, "Computation Beyond Turing Machines" and the entry on Interaction.

Within an operating system, the various kinds of interrupt signals are differentiated by an identifier, which is mapped to a short handler program by an *interrupt vector*. In this way, typing on a keyboard can be handled differently from a packet arriving over the network. The notion of an interrupt vector, however, can be rethought, not only in terms of how particular external events extend into actions within the operating system, but also in terms of how the actions of a particular piece of software are themselves extended into, and are extensions of, various sorts of social actions. The interrupt vector, then, becomes a carrier through which different elements of a social assemblage are associated. The social aspect of software unfolds in the very process of making these associations. Latour describes the social as being the associations that link different *actors* in time and space.[3] These actors can be humans, or non-human objects. An actor is any entity that plays a significant part in the formation of associations from which the social is formed. The interrupt is one principle through which such associations can be constructed and broken. During a lecture by the philosopher Jacques Derrida, a member of the audience, the cultural theorist Avital Ronell, interjected with the question: "How do you recognize that you are speaking to a living person?" to which Derrida responded: "By the fact that they interrupt you.".[4] In this sense, we could say that software's "cognition" of the social is comparable to Derrida's. Indeed, the action of interruption, of the break, is fundamental to the notion of the *gram*, the mark that differentiates, upon which Derrida's grammatology, the study of the role of inscription in the construction of human social and cultural systems, is based.[5] The interrupt, therefore, is the mechanism through which the social, as a process of making and breaking associations with others, is inscribed into a piece of running software.

If software is understood as an actor in such assemblages, then the operational space in which it performs is potentially the space of an entire assemblage, one which grows and contracts as circumstances change. The combinations in which software operates are often more complex than might first be assumed. A typical piece of desktop software, such as a text editor program, operates within an assemblage that includes not only the software itself and the user but also the operating system on which the program runs, and the devices through which the user interacts with it: the mouse, keyboard, and screen. If the keyboard is removed, the text editor program becomes inoperative, even though the program itself has not been altered. Elements such as the keyboard also provide a form of liminal boundary. When we press the keyboard we are literally and consciously entering into the operational space of the software. The situation becomes more complex, however, as we start to consider the kinds of assemblage that are constituted by other forms of software, such as those in embedded devices, and the *actors* with which they operate, such as radio frequency identification tags (RFID). Whereas we might describe the operational space of software in the context of a user at a desktop system as having a liminal boundary, these other, far more distributed, forms of software operate in a much more porous situation. Liminal boundaries are those that draw a distinct line, that one can have a definite sense of crossing, of being inside and outside of. Porous boundaries are less distinct; it is harder to tell when one is inside or outside, and they may have qualities of absorbency and leakage. Some assemblages may consist of multiple operational spaces, either nested or overlapping. The interrupt can therefore be thought of, on an extended level, as the vector that not only constructs associations between actors, but also traverses varying operational spaces.

Transport systems has been one of the main fields of deployment of such porous software systems. A combination of road surface sensor systems and networked CCTV cameras, linking in various analysis tools, have brought roadways into the operational space of software such as Automated Number Plate Recognition Systems (ANPRS) and Intelligent Transportation Systems (ITS). These systems monitor traffic flow for irregular incidents such as speed violation and breakdowns, or track vehicles in Congestion Charging Zones such as that in central London.[6] The roadway itself becomes a software interface, and road-markings and traffic signs all become actors within the assemblage of the roadway's operational space. The cars traveling on the roads may contribute their own software actors, in employing intelligent braking systems, or GPS navigation consoles. Within the process of airline travel, numerous software

[3] Bruno Latour, *Reassembling the Social: An Introduction to Actor-Network Theory*. Latour originally used the term *network*, as in his Actor-Network Theory, but has recently shifted to *assemblage*. The latter carries stronger connotations of something that is put together and taken apart and possibly quite contingent, which *network* does not convey. As it also helps keep a clearer conceptual distinction between a computer network and a social assemblage, I have used it here.

[4] The event is described in Avital Ronell, *Finitude's Score: Essays for the End of the Millennium*, p. 3.

[5] Jacques Derrida, *Of Grammatology*.

[6] For an overview of such systems, see Stephen Graham and Simon Marvin, *Telecommunications and the City: Electronic Spaces, Urban Places*. One of the key algorithms used in traffic analysis is the McMaster algorithm developed at McMaster University, Toronto; see Fred L. Hall, "McMaster Algorithm."

actors enter in and out of a variety of assemblages that travelers, pilots, and other staff all, similarly, enter and exit. These include the software that manages the transport of luggage and tourists through the airport terminal, the software that analyzes x-ray scans of luggage, the passport systems that log traveler IDs, which, in turn, are often connected to automated photographic devices or biometric scanners. The interoperability of runway markings, air corridors, and control tower navigation systems, and the on-board flight controllers also play a part. On an average day, an individual in a city may connect and disconnect from numerous assemblages involving different software actors. Frequently,they are unaware of the various operational spaces that they have interrupted: using mobile phones, smart cards on public transport systems (such as London's *Oyster card*), bank autoteller machines, RFID tagged goods, or a keycode to access a building. The CCTV system of a bank, office, or housing estate may be linked up to movement analysis tools, seeking to detect a possible hold-up scenario, or irregular movement patterns among the building's occupants. The introduction of chip-carrying biometric identity cards, as is currently planned in the United Kingdom, may bring with it the ability to cross-reference these cards and the readings of CCTV facial analysis systems, linking the interruptions of human activity in urban space to singular identities, just as logging onto a computer links the interrupts of keyboard and mouse to a particular username.[7] The operational space of software extends over large physical areas in which algorithms become the arbiters of normative behavior and of inclusion and exclusion. The *Cartografiando el Territorio Madiaq* is an ongoing project to map the complex of surveillance systems, military bases, and communication infrastructures that are in place across the Strait of Gibraltar between Spain and Morocco.[8] It demonstrates the complex assemblages of actors (technological, military, and legal) that are involved in policing the Spanish borders. The play of the liminal and porous in evidence here is not only one of boundaries along the operational spaces of various software systems, but also the construction of the European Union's own political and economic boundaries which, through such surveillance, become conflated with software processes.

Porous is not the same as open. A porous surface acts as a regulatory mechanism, as the porosity of skin regulates the flow of moisture and air between the body and its environment. The systems described above create porosity in otherwise open spaces. The regulatory trajectory, however, is not exclusively one-way. In a memoir, one of the inventors of the interrupt mechanism, Edsger Dijkstra, wrote:

> It was a great invention, but also a Box of Pandora. Because the exact moments of the interrupts were unpredictable and outside our control, the interrupt mechanism turned the computer into a nondeterministic machine with a non-reproducible behavior, and could we control such a beast?[9]

The interrupt increases the contingency of the environment in which a piece of software runs. In constructing associations with an *outside* it makes the operation of software more situated in that outside and, therefore, prone to the contingencies of that outside environment.[10] The interrupt transfers governance back and forth between computer and user, or other outside actors. Around every piece of software, a set of shadow practices develop that are not inscribed in the code itself, but on which its ability to act depends. Christian Heath and Paul Luff's studies of the use of software in businesses and organizations demonstrates that the software is often only effective when nested within larger structures of governance that guide the gestures of those who interact with it.[11] This combined governance of software and user environments is sharply evident in call centers, in which a hybrid software and managerial

[7] Computer vision and video analysis is currently a major area in computer sciences research; example projects include: Jaime Dever, Niels da Vitoria Lobo, and Mubarak Shah, "Automatic Visual Recognition of Armed Robbery," and Douglas Ayers and Mubarak Shah, "Monitoring Human Behavior from Video Taken in an Office Environment." Proponents of algorithmic-based facial recognition often state that it does not suffer from problems of social and cultural prejudice that human surveillance staff often bring with them. Studies of such algorithms, however, have shown that due either to the data sets on which they are trained, or empirical factors in how they operate, they may still demonstrate aspects of differential treatment for different ethnic groups, which result in racially-weighted responses; see Lucas D. Introna and David Wood, "Picturing Algorithmic Surveillance: The Politics of Facial Recognition Systems."

[8] Hackitectura, *MAPA: Cartografiando el territorio madiaq*.

[9] Edsger W. Dijkstra, "My Recollections of Operating System Design," pp. 13 – 14. In this document, Dijkstra also discusses some of the limitations of the polling method.

[10] Lucy Suchman has analyzed this aspect of computer use in detail through the concept of "situatedness" in Lucy Suchman, *Plans and Situated Actions: The Problems of Human-Machine Communication*.

[11] Christian Heath and Paul Luff, *Technology in Action*.

infrastructure maintains the overall mechanism.[12] What might be called "counter-interruptive" practices also develop, such as maps of CCTV and traffic cameras enabling people to plan routes that avoid them, or call center employees who trigger fake systems crashes to buy a bit of unlogged free time.[13] The transfer of governance can also be an opportunity to interrupt its initial vector and claim other possibilities.

If the interrupt teaches us anything about software, it is that software is in many cases only as effective as the people who use it, those nondeterministic machines with their complex, non-reproducible behaviors, those *others* on whom it relies — can it really control such beasts? To understand software in terms of the interrupt is to understand it in terms of its place within larger structures of social formation and governance. Software engineering is simultaneously social engineering. Software criticism, therefore, must also be simultaneously social. In critically engaging with software, we must not only map the vectors of the interrupt, but also seek to make our own interruptions, to pose questions and insert alternative vectors and practices within the assemblages it connects to.

[12] A detailed account of various employees' experiences of working in call centers is provided in Kolinko, *Hotlines: Call Centre Inquiry Communism*.

[13] The Institute for Applied Autonomy's *iSee* is an interactive online map of CCTV systems in central New York; it enables users to plot a path of least surveillance between two locations in the city: http://www.appliedautonomy.com/isee.html. http: //www.controleradar.org is a French website providing listings of computer-controlled road cameras across France. Kolinko, *ibid.*, provides several accounts of ways in which call center employees have tried to counteract the conditions under which they work.