/>11

L'evoluzione dei sistemi operativi

Carlo Spinedi
Amos Brocco
ISIN, Istituto sistemi informativi e networking, DTI, SUPSI

Fra le definizioni che si possono leggere per "sistema operativo" (s.o.) troviamo che è un *software che permette all'utente di usare convenientemente il computer*. Molta gente tende ad identificare il s.o. con la sua interfaccia. Anche parecchi articoli divulgativi che si possono leggere nelle riviste all'apparire di nuove versioni dei più popolari s.o. si concentrano per lo più sul modo di interagire dell'uomo con gli oggetti presenti sul *display* o sull'aspetto e la presenza di determinate icone o menu. Da quando sono apparsi *tablet* e *smartphone* il modo di interagire con il computer è notevolmente cambiato: la tastiera ha ceduto il posto al *touch screen* e all'interprete vocale. Ma l'interfaccia è solo ciò che si vede: i moduli presenti in un s.o. sono parecchi e il loro sviluppo è un processo che dura da più di cinquant'anni. Il fatto che ci sia "ben altro" oltre ad un'interfaccia è chiarito da un'altra definizione di sistema operativo, considerandolo come un'entità nascosta che permette ai programmi di funzionare sui nostri computer: l'utente non utilizza un sistema operativo, ma utilizza piuttosto degli applicativi che possono essere eseguiti grazie alla presenza di un sistema operativo.

/articolo-11/> capitolo-1

Primi computer senza s.o.

I computer costruiti negli anni '40 e '50 del secolo scorso erano privi di s.o. Il computer eseguiva un programma per volta per un singolo utente. I programmi erano caricati mediante sequenze di operazioni manuali o di istruzioni elementari oppure a partire da supporti di memoria quali schede o nastri di carta perforati e magnetici precedentemente preparati con apparecchiature separate. L'ENIAC (*Electronic Numerical Integrator and Computer*), realizzato verso il 1945 all'Università della Pennsylvania per risolvere problemi balistici, era programmato disponendo correttamente una moltitudine di interruttori e realizzando una quantità di collegamenti elettrici mediante cavi mobili.

La preparazione di un programma richiedeva a un team di cinque persone alcuni giorni di lavoro e la ricerca di eventuali errori di programmazione (i cosiddetti *bug*) poteva costare altri giorni di lavoro. La programmazione divenne più rapida con l'introduzione degli interruttori telecomandati, la cui posizione, memorizzata precedentemente su un nastro perforato mediante una *teletype* veniva impostata automaticamente durante la lettura del nastro.

L'EDVAC (*Electronic Discrete Variable Automatic Computer*) e l'EDSAC (*Electronic Delay Storage Automatic Calculator*) realizzati attorno al 1949, sempre all'Università della Pennsylvania, sono fra i primi computer dove il programma non deve più essere codificato in forma rigida mediante interruttori e cavi, ma può essere caricato, prima di essere eseguito, in una memoria dinamica. Si parla già di programma, ma non ancora di linguaggio di programmazione.

È il BINAC (*Binary Automatic Computer*), successore commercializzato dell'ENIAC, a introdurre nel 1949 una rudimentale codifica di programmazione, il *Short Order Code* (SOC), ideato da John Mauchly. Il SOC, che fa uso di un semplice interprete algebrico, è un antenato dei successivi linguaggi di programmazione. Durante la preparazione di questi primitivi programmi ci si è resi presto conto che ogni volta si doveva ripetere la programmazione delle stesse attività, come ad esempio l'introduzione dei dati o la stampa dei risultati. Furono perciò inventate le *subroutine*, cioè le funzioni di base da richiamare per eseguire un determinato compito. Queste funzioni, raccolte in librerie, hanno formato l'*Input/Output Control system*, che può essere considerato una forma primitiva di s.o.

Il s.o. non è stato indispensabile per molti pionieri dell'informatica: ancora nel 1961, al MIT Ivan Sutherland ha sviluppato un programma per la grafica interattiva con un'interfaccia molto evoluta, lo Sketchpad. Il programma è stato scritto in assembler su un computer sperimentale transistorizzato, ma privo di s.o.

/articolo-11/> capitolo-2

Mainframe e s.o. batch:

I grandi costi dei primi computer commercializzati negli anni '50 richiesero di sfruttare al massimo le loro risorse di calcolo minimizzando i tempi morti necessari per la preparazione del programma da eseguire. Venne inventato un metodo di lavoro che consentiva di preparare il computer e per l'esecuzione del prossimo programma già durante l'esecuzione di quello precedente. L'elaboratore scientifico IBM 704 fu fra i primi computer dotati nel 1956 di un s.o. *single job batch* (s.o. a lotti monoprogrammato), software sviluppato nel laboratori di ricerca della General Motors (GM-NAA I/O). I programmi, detti *jobs*, memorizzati in pacchetti di schede di carta perforate, erano letti meccanicamente e le schede di controllo poste all'inizio o intercalate nei pacchetti informavano il s.o sul come interpretare il contenuto delle schede seguenti, distinguendo se si trattava di codice di programma oppure di dati.

I s.o. batch monoprogrammati eseguivano un *job* dopo l'altro. Per introdurre i dati o per memorizzare risultati o informazioni temporaneamente durante l'esecuzione, oltre alle schede perforate venivano usati sovente nastri magnetici. I dispositivi per l'input/output erano però apparecchiature lente a confronto dell'unità centrale del computer e quindi i tempi d'attesa, durante i quali il processore doveva aspettare per ricevere o fornire dati, erano lunghi. Si passò così progressivamente ai s.o. batch in grado di eseguite più job simultaneamente. I tempi di attesa di input/output per un determinato job venivano sfruttati per elaborarne un altro: si parla ora di s.o. batch multi programmato. L'elaboratore inglese Leo III, fu uno dei primi ad essere dotato di un tale sistema.

Figura 1
Un *Midrange Computer* della fine degli anni'60: il PDP 8

/articolo-11/> capitolo-3

Il s.o. interattivo

In un s.o. batch non è previsto di poter influenzare l'esecuzione in corso da parte dell'utente. Ogni errore fatale riscontrato nelle fasi di compilazione o di esecuzione implica la fine del processo e quindi all'utente non resta che sottoporre di nuovo l'intera esecuzione del job dopo l'identificazione e la rimozione della causa dell'errore in base alle informazioni fornite dal sistema. Questa modalità di lavoro, che richiedeva lunghi tempi per sviluppare nuovi programmi, ha indotto a sviluppare s.o. che permettessero maggiore interattività con l'utente e in particolare la possibilità di poter intervenire durante le fasi di esecuzione di un job, di apportare velocemente modifiche al codice del programma e riprendere rapidamente il processo. Il successo dei cosiddetti *midrange computer* (minicomputer a costi contenuti), nati all'inizio degli anni '60 del secolo scorso, sta nel fatto di aver attribuito fin dall'inizio grande importanza alle possibilità di interagire con i programmi e con il sistema.

Su grossi sistemi, la necessità di ottimizzare le risorse, ha portato invece a offrire a più utenti la possibilità di interagire contemporaneamente con il computer tramite i s.o. di tipo *time-sharing*. In questo caso l'esecuzione di più job era portata avanti suddividendo il tempo di calcolo disponibile in parti uguali fra gli utenti. Il *Compatible Time-Sharing System* (CTSS), presentato nel 1961 al MIT è uno dei primi. Fu detto compatibile in quando doveva garantire la compatibilità con sistemi batch precedenti dell'IBM. Gli utenti interagivano con il sistema mediante telescrivente. Fra i progetti di s.o. di tipo *time-sharing* vale la pena di ricordare il sistema MULTICS (*Multiplexed Information and Computing Service*) sviluppato a partire dal 1965 al MIT. Dal 1969 fu impiegato, sia nelle università che nelle aziende, per ospitare sistemi informativi. Si diffuse non solo in America, ma anche in Europa.

/articolo-11/> capitolo-4

I s.o multitasking

Con il minicomputer i s.o. interattivi evolvono con l'introduzione del cosiddetto s.o. *multitasking*. Si tratta di un s.o. che permette l'esecuzione apparentemente simultanea di più programmi dove l'utente ha continuamente il controllo dei programmi che sta eseguendo. Per gestire l'avanzamento dell'esecuzione, diversi algoritmi sono stati sviluppati. Quelli più diffusi oggi si basano sui concetti di priorità e di quanto di tempo. Il processore lavora per il programma a priorità più alta (stabilita dall'utente o dal s.o. stesso) e a parità di priorità il s.o. concede ad ognuno l'uso del processore durante un quanto di tempo (pochi millesimi di secondo). Quando tutti gli programmi in attesa di essere eseguiti avranno consumato il loro quanto, il primo potrà di nuovo essere eseguito. (Per correttezza: oggi non si parla più di programmi, ma di processi e di *threads*).

/articolo-11/> capitolo-5

I s.o. per la multiutenza

Si è già visto che i s.o. di tipo *time-sharing* sono già stati creati per essere usati contemporaneamente da più utenti. L'introduzione della multiutenza ha rappresentato per i progettisti una grande sfida in particolare per quanto attiene agli aspetti di sicurezza che la condivisione di dati e programmi di utenti diversi su uno stesso sistema comporta. Un s.o. per la multiutenza deve essere in grado di concedere i permessi di accesso ai documenti memorizzati in modo differenziato (solo lettura, modifica, esecuzione, ecc.) a dipendenza dell'utente che fa richiesta del permesso (proprietario del documento, utente autorizzato, amministratore, ecc.).

Il sistema deve inoltre garantire la completa confidenzialità del lavoro svolto da un programma nei confronti di altri programmi attivi e inoltre non deve permettere all'esecutore di un programma di accedere a informazioni sensibili dimenticate da altri utenti in spazi di memoria temporanea. Per permettere di mettere a confronto s.o. in base alla loro sicurezza, a partire dagli anni '80 sono stati definiti i criteri necessari per collocare un s.o. in una determinata classe di sicurezza (vedi TCSET, ITSEC, Common Criteria). Nel corso degli anni, i requisiti di sicurezza hanno influenzato anche lo sviluppo dei processori introducendo ad esempio differenti modalità di lavoro (user mode, kernel mode, ecc.) più o meno privilegiate. Oggi, i comuni s.o. sono in grado di garantire la confidenzialità, ma è necessario che gli utenti conoscano i principi su cui si basa e rispettino le raccomandazioni contenute nelle guide all'uso.

/articolo-11/> capitolo-6

Alcune astrazioni di successo

Lo sviluppo dei s.o. nel corso degli anni è associato all'introduzione di alcuni importanti concetti che hanno facilitato l'utilizzo dei computer nonché lo sviluppo dei programmi applicativi. Il concetto che si può probabilmente considerare come il più importante nel mondo dell'informatica è senza dubbio quello del *file* e della *directory*, che insieme costituiscono l'astrazione che è conosciuta come *filesystem*. Un *filesystem* offre la possibilità di gestire i propri dati in maniera del tutto simile alla realtà, senza preoccuparsi di come i dati sono codificati sul dispositivo di memorizzazione, separando le informazioni in documenti che possono essere organizzati gerarchicamente in cartelle e che possono essere facilmente condivisi.

Un'altra astrazione, quella della memoria virtuale, ha permesso di scrivere programmi senza doversi occupare della gestione della memoria fisica del sistema (quanta memoria fisica è disponibile o quali locazioni sono libere). Ogni programma ha a disposizione la stessa quantità di memoria contigua, limitata solo dal numero di bit utilizzati dal processore per l'indirizzamento (8, 16, 32, 64). Il compito di mappare questa memoria virtuale con quella fisica è lasciato al s.o. e in particolare alla sue componenti per il *memory management*.

Anche il concetto di *driver* è un'astrazione di successo. I s.o. devono poter accedere agli innumerevoli dispositivi periferici conosciuti o che saranno sviluppati più tardi. Il *driver* è la componente software del s.o. che permette l'accesso a dispositivi, talvolta anche molto diversi fra di loro, con le medesime modalità e funzioni. Infine i cosiddetti servizi di sistema, con cui un programma fa richiesta di informazioni o chiede l' accesso a una determinata risorsa, permettono di operare in modo sicuro e affidabile, poiché è il s.o. che svolge il lavoro sensibile e non il programma.

/articolo-11/> capitolo-7

La portabilità dei s.o.

Il rapido susseguirsi di nuove architetture hardware ha posto i progettisti di s.o. davanti alla sfida di realizzare in tempo utile il software necessario per il loro sfruttamento. Il rischio di produrre s.o. che sarebbero diventati in pochi mesi obsoleti poiché concepiti solo per una determinata architettura hardware era evidente. I creatori del s.o. Unix, sviluppato nei laboratori Bell attorno al 1969, sono stati pionieri nell'aver attribuito grande importanza al fatto di poter usare il sistema non solo con i processori fino ad allora conosciuti, ma anche con i futuri. Due sono stati i principali accorgimenti adottati: concentrare tutte le funzioni che interfacciavano direttamente l'hardware in un'unica zona di programma e, dove possibile, usare per programmare il s.o. un linguaggio che non avesse nessuna stretta relazione con una determinata architettura hardware. Nacque allora il linguaggio di programmazione C. È stato più volte affermato che uno dei motivi per cui il s.o. Unix si è imposto è la sua facilità nell'essere trasportato: ancora oggi Unix è il primo e sovente l'unico s.o. che viene installato su un nuovo modello di supercomputer. Nel corso degli anni altri s.o proprietari sono stati creati o riscritti per essere agevolmente trasportati su nuove architetture.

/articolo-11/> capitolo-8

Dalla shell testuale a quella grafica

Con i primi s.o. interattivi sviluppati a partire dagli anni '60 l'utente interagiva mediante una telescrivente provvista di tastiera. Più tardi apparvero i videoterminali dotati di monitor monocromatici alfanumerici (visualizzavano solo testo e numeri). Per permettere di operare con un s.o interattivo, fu necessario sviluppare insiemi di comandi verbali da impartire con la tastiera. Nei s.o della famiglia Unix i comandi sono spartani, fanno capo sovente ad acronimi e non sono certo un modello di eleganza sintattica. Altri sistemi per lo più proprietari hanno introdotto veri e propri linguaggi di comando con regole sintattiche precise come ad esempio nel caso del *Digital Control Language*. I linguaggi di comando sono importanti anche ai giorni nostri, poiché servono a creare le procedure da eseguire nella modalità batch, funzionalità preziosa anche nei moderni computer

interattivi. Occorre attendere fino all'apparizione del video grafico dove è possibile il controllo immediato di ogni suo pixel per vedere nascere, nella metà degli anni '70 le prime *shell* grafiche. Un'interfaccia grafica necessita però anche di un puntatore con cui l'uomo può indicare al sistema a quale *pixel* o a quale area grafica vuol fare riferimento. Uno degli strumenti più antichi sviluppati a questo scopo è la penna ottica nata al MIT attorno al 1952. Alcuni anni dopo, nello stesso istituto viene inventato il *joystick*, ma lo strumento che più ha successo è il *mouse*, che nella forma che conosciamo, viene presentato per la prima volta a Standford nel 1968 (occorre però osservare che già nel 1947 uno strumento con la stessa funzionalità fu realizzato, ma a quei tempi con i computer interattivi non esistevano).

All'interfaccia grafica sono interessati sia i produttori di costosi computer dedicati alla progettazione e al disegno di macchine, edifici, ecc., sia gli ideatori di computer personali che fanno la loro apparizione all'inizio degli anni '80. Ci si rende però conto che la gestione di schermate grafiche ricche di finestre e icone richiede computer più potenti e dotati di molta più memoria che in passato. La nascita della shell grafica stimola perciò i produttori a sviluppare hardware più performanti.

Durante gli anni '80, s.o. già affermati, ma dotati solo di shell verbale, integrano la famosa interfaccia grafica X Window sviluppata al MIT. Questa *shell* è particolarmente interessante, perché può operare a distanza attraverso la rete e permette all'elaboratore centrale precedente, collegato solo a videoterminali alfanumerici, una naturale evoluzione. Questa funzionalità è stata adottata molti anni dopo anche nelle reti di computer personali pensati all'inizio per lavorare in modo isolato.

/articolo-11/> capitolo-9

La comunicazione fra s.o.

Fin dalla nascita dei primi *mainframe*, l'esigenza di far comunicare computer fra di loro e con dispositivi periferici è stata presente. A partire dagli anni '50 e durante un ventennio, si è trattato principalmente di collegare grossi elaboratori alle stazioni di controllo (videoterminali) attraverso linee telefoniche o cavi coassiali. Le soluzioni adottate e i protocolli di comunicazione erano proprietari e l'esigenza di far comunicare sistemi operativi fra di loro non era ancora molto sentita. Alla metà degli anni '70 si assiste alla nascita delle prime vere e proprie reti di computer dove è possibile lo scambio di documenti o l'esecuzione di programmi su sistemi remoti. Queste reti sono ancora basate su protocolli proprietari che già in quegli anni avevano raggiunto un alto grado di sofisticazione. Famose rimangono le reti basate sul protocollo DECNET.

I protocolli attuali usati in internet per la comunicazione fra s.o. hanno le loro radici in un progetto del Dipartimento della difesa degli Stati Uniti iniziato nel 1969. La rete ARPAnet realizzata fra quattro università (UCLA,UC Santa Barbara, Stanford e l'Università dello Utah) ha permesso di sperimentare le prime versioni delle tecnologie telematiche che porteranno, una

quindicina di anni dopo, all'introduzione di TCP/IP il principale protocollo di internet che ha permesso una rapida espansione della rete in tutto il mondo. L'arrivo di questo protocollo, alla cui conoscenza ha contribuito la rete stessa, ha condannato in pochi anni tutti i protocolli proprietari precedenti all'oblio, anche se talvolta ben più raffinati degli standard che si sono imposti. Il collegamento dei computer alle reti ha però creato nuovi problemi per quanto concerne la sicurezza del s.o.: ancora oggi non mancano progetti per migliorare la sicurezza e confidenzialità dei sistemi in rete.

/articolo-11/> capitolo-10

I s.o. aperti

I principali s.o. che si sono imposti sul mercato fino all'inizio degli anni '90 sono sistemi proprietari e quindi brevettati e possono essere usati solo su licenza. La loro codifica non è resa pubblica e quindi non è possibile apportare modifiche senza esplicito permesso del proprietario del marchio. L'impossibilità di modificare e condividere liberamente il codice sorgente spinse Richard Stallman, allora ricercatore presso il MIT, a creare il progetto GNU (GNU's Not Unix, ovvero GNU non è Unix). Questo progetto, nato nel 1984, aveva lo scopo di creare un sistema operativo libero e completo ispirato a Unix. Grazie a GNU vedono la luce diverse versioni libere di programmi tipici degli ambienti Unix, come editor di testo e compilatori. Grazie al rilascio di Linux come software libero, nel 1992, il progetto si arricchisce di una componente fondamentale, il *kernel*. Importante è il fatto che il codice viene reso pubblico con lo scopo di stimolare una comunità di interessati a contribuire al suo ulteriore sviluppo. I sistemi GNU/Linux vogliono essere compatibili, per quel che riguarda l'esecuzione di programma, con i principali sistemi Unix precedenti. A partire da quell'anno, chiunque fosse interessato può disporre del codice sorgente completo di un s.o., lo può studiare, modificare e usare liberamente. Il successo del progetto GNU, del *kernel* Linux e del software libero sono documentati dalla forte crescita di apparecchiature che sono state dotate di questi software durante gli ultimi 25 anni.

/articolo-11/> capitolo-11

La virtualizzazione e il cloud

Negli ultimi 20 anni, la necessità per le aziende di dotarsi di un cospicuo numero di sistemi dedicati ad applicazioni diverse ha portato a sviluppare tecnologie che permettono di eseguire sullo stesso computer più s.o. contemporaneamente. Questa innovazione, che si fa strada dapprima sui grossi sistemi e che permette però solo l'attivazione di più istanze dello stesso s.o, evolve verso la cosiddetta visualizzazione dell'hardware, dove sul computer viene dapprima installato un programma di controllo che crea e simula determinati ambienti hardware (le macchine virtuali) che permettono a loro volta di eseguire più istanze di s.o. anche

diversi fra di loro. La virtualizzazione ha permesso anche la nascita del *utility computing* o on *demand computing*, ovvero un modello di servizi informatici in cui il fornitore offre delle risorse computazionali o di memorizzazione dei dati *a richiesta*. Questo modello di business permette una migliore gestione delle risorse e degli investimenti, ed è alla base del concetto di *cloud*, in cui l'infrastruttura, la rete e le applicazioni diventano dei servizi gestiti da terzi, acquistabili in base alle proprie esigenze.

/articolo-11/> capitolo-12

I più popolari s.o. odierni

La quasi totalità dei computer usati oggi, sia personali che aziendali, operano con s.o., le cui origini risalgono agli inizi degli anni '70, agli anni dove negli Stati Uniti la ricerca ha dato un forte contributo allo sviluppo nell'informatica. I principali filoni sono due: da una parte quello basato sullo stile introdotto da Unix, e che ha portato non solo a Linux ma ad esempio anche al s.o. prodotto dalla Apple, dall'altra quello confluito negli attuali sistemi s.o. Windows della Microsoft, che ha avuto importanti predecessori sviluppati all'inizio per i minicomputer a 32 e 64 bit, quali OpenVMS che, a sua volta, trae le sue origini dai s.o. operanti sui processori PDP 11 a 16 bit quali RSX11 della DEC. Nell'ambito dei sistemi operativi per dispositivi mobili, quali *smartphone* e *tablet*, oggi sono i sistemi operativi Android (basato su Linux) e iOS a contendersi il mercato, dopo essere riusciti a scavalcare concorrenti quali Symbian OS e Windows Mobile, che fino agli inizi di questo decennio detenevano ancora importanti quote di utilizzatori.

/articolo-11/> capitolo-13

Sistemi invisibili

L'evoluzione tecnologica, la miniaturizzazione delle componenti hardware e il conseguente aumento delle prestazioni hanno portato i sistemi operativi in diversi ambiti della nostra vita quotidiana. Dai televisori, ai frigoriferi, ai sistemi di controllo industriali o di un'automobile, sono molti i dispositivi che oggi dipendono da un sistema operativo *invisibile* per il loro funzionamento: si parla in questo caso di sistemi operativi *embedded* (o imbarcati). Questi sistemi hanno dei requisiti diversi rispetto a un computer tradizionale: non devono gestire applicativi come editor di testo o un browser Internet, spesso non si interfacciano neppure con un utente. Al contrario essi devono essere affidabili, efficienti e in grado di funzionare senza nessun tipo di supervisione e con risorse limitate. Molto spesso un sistema operativo *embedded* deve garantire lo svolgimento di compiti con tempi di risposta prevedibili, in quelli che sono definiti come requisiti *real-time*: il risultato di un'operazione eseguita dal sistema non deve essere solo corretto dal punto di vista logico e computazionale, ma deve essere fornito entro un certo intervallo di tempo.

/articolo-11/> capitolo-14

Il futuro dei sistemi operativi

Nonostante i sistemi operativi correnti abbiano raggiunto dei livelli di funzionalità e di semplicità di utilizzo elevati, la ricerca nel campo è comunque attiva su più fronti. In particolare troviamo da un lato i problemi legati alla gestione ottimale delle risorse di calcolo parallele e distribuite, dall'altro la gestione di dispositivi legati al mondo dell'*Internet of Things* (IoT), in cui efficienza energetica, sicurezza e interoperabilità rappresentano gli aspetti più importanti. Dal punto di vista degli utenti, i sistemi operativi continuano ad evolversi di pari passo con l'evoluzione tecnologica, per supportare nuove tecnologie e per proporre nuovi paradigmi di interazione.