1 Lezione del 24-09-25

1.1 Cenni storici

Le prime macchine calcolatrici "moderne" nascono durante la seconda guerra mondiale, principalmente per scopi crittografici.

Fu nel periodo del secondo dopoguerra che diverse industrie, principalmente dal settore delle macchine da scrivere e di apparecchiature simili, decisero di sviluppare queste tecnologie per scopi di ricerca e commerciali.

Di pari passo diverse università iniziarono a loro volta a sviluppare architetture e macchine calcolatrici, in questo caso a puro scopo di ricerca. Un esempio locale è quello della **CEP** (*Calcolatrice Elettronica Pisana*), sviluppata dai dipartimenti di matematica e fisica di Pisa (sotto indicazione di Enrico Fermi) per aiutare i ricercatori nei loro calcoli.

Sempre a Pisa fu l'ingegnere Mario Tchou a lanciare, in collaborazione con Olivetti, il progetto che diventò nel 1959 l'**Elea 9003**, fra i primi calcolatori a transistor commerciali (di contro la CEP funzionava a valvole termoioniche).

1.1.1 Sistemi Batch

In queste prime macchine, anche se la possibilità della multiprogrammazione era disponibile, raramente si parlava di "sistemi operativi" veri e propri. I primi sistemi operativi nascono quindi per i mainframe degli anni '60, fra cui notiamo gli **IBM Sistema 360** (e i successivi Sistema 370).

Inizialmente, queste macchine venivano usate in modalità **batch** (più programmi di più utenti eseguiti in sequenza): i primi S/O nascono appunto per permettere l'uso simultaneo (*time-sharing*) della macchina da parte di più utenti.

In ogni caso, già nei primi sistemi batch monoprogrammati si necessitava di diversi componenti effettivamente assimilabili ad un rudimentale sistema operativo:

- Un sistema di programmazione in memoria di massa (all'epoca nastri magnetici);
- Una *Job Control Language* (**JCL**), che esprimeva direttive interpretate da un *Monitor* (antenato delle moderne *shell*);
- Un **BIOS** (*Basic Input Output System*), cioè un insieme di routine per l'interazione con le periferiche.

L'S/O era quindi composto da Monitor + BIOS, che poteva essere configurato per caricare programmi e mandarli in esecuzione. In ogni momento in memoria si trovavano comunque il S/O e al più un programma utente.

1.1.2 Sistemi di spooling

Il prossimo passo è quello dei sistemi di **spooling** (*Simultaneous Peripheral Operation On-Line*). Questi nascono per permettere al programma utente di restare in esecuzione mentre le periferiche (all'epoca molto lente) completano le loro operazioni, bufferrizzando quindi le operazioni di ingresso/uscita.

I sistemi operativi che implementavano lo spooling dovevano quindi arricchirsi per permettere questo tipo di funzionalità.

1.1.3 Sistemi multiprogrammati

Arriviamo quindi ai sistemi **multiprogrammati**, cioè che permettono la gestione contemporanea di più programmi nella memoria principale: per la prima volta oltre al sistema operativo possiamo caricare in memoria più di un singolo programma utente.

I sistemi operativi di questo tipo si dovranno quindi dotare di diverse funzionalità, fra cui *scheduling* dei processi, possibilità di fare **DMA** (*Direct Memory Access*) sulle periferiche, *preemption* dei programmi in esecuzione, *memoria virtuale* per permettere mappature in memoria localmente costanti per ogni programma, ecc...

1.1.4 Sistemi time-sharing

Lo sviluppo di sistemi di tipo multiprogammato è stato favorito dal fatto che i programmi utente che venivano sviluppati erano sempre più *interattivi*, quindi caratterizzati da fasi temporali distinte:

- **CPU-Burst**, dove il processore lavorava effettivamente sui dati;
- **I/O-Burst**, dove il processore attendeva operazioni I/O dalle periferiche, magari fornendosi del DMA.

Ci spostiamo quindi da un paradigma di esecuzione *sequenziale* ad un paradigma *multi-tasking*, dove il sistema operativo assegna ciclicamente istanti temporali (*quantum*) ai processi in esecuzione.

Il vantaggio dell'esecuzione multitasking è di poter avvicinare fra di loro i CPU-Burst, spostando il controllo della CPU da un processo all'altro quando si incorre in un I/O-Burst.

Per quanto ci riguarda, quindi, la tecnica del **time-sharing** non è che un modo per implementare il *multi-tasking*, cioè un caso particolare della *multiprogrammazione*, caratterizzato da processi in memoria che vengono eseguiti (o almeno hanno l'illusione di essere eseguiti) contemporaneamente. Ricordiamo che l'esistenza di più processi in memoria era di per sé caratteristica del sistema multiprogrammato.

L'idea di sviluppare diversi e sofisticati algoritmi di *scheduling* viene proprio dalla necessità di dover mantenere la CPU in piena attività, cioé eseguire più CPU-Burst possibile, scegliendo in maniera intelligente quali processi mandare in esecuzione (equivalentemente, a quali processi assegnare i quantum temporali).

Notiamo che il tempo che la CPU passa a realizzare lo scheduling e i cambi di contesto rappresenta effettivamente **overhead** per il sistema, cioè tempo non passato ad eseguire programmi utente, ma in qualche modo "sprecato" in altri modi. Questo overhead è giustificato solo nel caso in cui le virtualizzazioni che consente permettono una velocizzazione considerevole della macchina.

1.1.5 Sistemi in tempo reale

La storia dei sistemi operativi ha un'interessante tangente nei cosiddetti sistemi **real-time** (*in tempo reale*). Questi sono sistemi dove lo scheduling è *deterministico* e il tempo impiegato ad eseguire un dato processo può quindi essere stabilito prima che questo venga lanciato.

Sistemi di questo tipo sono utili nel caso di calcolatori che interagiscono con *ambienti operativi* reali attraverso **sensori** ed **attuatori**, dove la precisione temporale con cui vengono eseguite certe operazioni è effettivamente importante alla funzione della macchina.