# M1 - ARCHITETTURA DEL CALCOLATORE

## UD1 - ARCHITETTURA E FUNZIONAMENTO DEI SISTEMI DI ELABORAZIONE

### **Descrivere il principio di funzionamento del processore.**

Il processore è un dispositivo hardware in grado di eseguire istruzioni, a partire da un instruction set. E’ in grado di eseguire una istruzioni dopo l’altra (in maniera sequenziale) grazie al Program Counter, un registro che individua le parole di memoria desiderate.

Ad ogni istruzione, il processore esegue 3 fasi:

* **Fetch**, acquisizione dell’istruzione e caricamento della stessa nel registro Current Instruction Registry
* **Decode**, l’istruzione corrente viene decodificata ed analizzata (se è una istruzione valida) dalla unità di controllo Control Unit
* **Execute**, se l’istruzione è valida, la CU attiva l'unità aritmetico logica ALU (Arithmetic Logic Unit) in grado di provvedere all'esecuzione dell'istruzione e permette al processore di accedere alle parti di memoria necessarie per prelevare le informazioni necessarie per l'esecuzione dell'istruzione.

Il processore esegue in modo ciclico all’infinito le tre fasi di Fetch, Decode ed Execute in modo sequenziale. In caso di programmi complessi si usano spesso delle frasi condizionali o dei cicli per rieseguire un blocco di operazioni.

### **Descrivere come avviene la chiamata di procedura e la risposta all'interruzione.**

Una procedura ha un certo numero di parametri formali, e una chiamata di procedura è l'atto di chiamare questa procedura con certi valori dei parametri formali. Per permettere l'esistenza di una procedura, è necessario impostare una struttura dati (Stack, situata nella memoria centrale e al suo interno troviamo tutte le informazioni necessarie alla procedura) per gestire la chiamata di una funzione.

Quando viene chiamata una procedura, salvo relativo alla procedura attuale (PC, valori dei registri, parametri ed ecc...) in cima allo stack. Poi, sempre nello stack trovo dello spazio per contenere i valori della procedura: il suo PC, i suoi parametri etc. In questo modo, conoscendo l'indirizzo in memoria della cima dello stack, il codice della procedura saprà con certezza che, in modo relativo alla cima dello stack, il primo parametro si troverà 1 word dopo, il secondo 2 word dopo, l'indirizzo a cui impostare il PC alla fine della procedura in un certo altro posto e così via. In questo modo, creo il contesto di attivazione della procedura, eseguo il suo codice, e sono in grado di tornare poi al codice che ha chiamato la procedura. Ovviamente una procedura può chiamarne un'altra etc. etc.

Nel contesto di attivazione viene salvato anche l'indirizzo di partenza della porzione di stack che riguarda chi ha chiamato quella procedura. Questo serve per recuperare le variabili. Infatti, le variabili di una procedura vengono cercate prima nella porzione di stack a me allocata, poi se non le trovo vado nella porzione di stack relativa a chiamato me e le cerco lì, se non le trovo ancora vado nella porzione di stack relativa a chi ha chiamato il chiamante etc. etc.

Le variabili locali scompaiono quando la procedura termina: non servono più, e lo spazio che occupavano deve essere reso nuovamente disponibile. Alla fine della procedura, poi, si ripristina il contesto di chi l'aveva chiamata: PC, registri e così via.

La differenza tra chiamata di procedura e risposta ad interrupt è la sincronicità. Una procedura viene sempre chiamata con una CALL. La risposta ad un interrupt invece no, non è predicibile quando avverrà (asincrona rispetto al processore).

Quando si risponde ad un'interruzione, si salva tutto il contesto locale e si salta ad un indirizzo che era stato precedentemente associato a quella particolare interruzione. Si esegue il codice di quell'indirizzo, e alla fine si ritorna. Siccome non c'è nessuna CALL, durante la risposta ad un interrupt in genere si disabilitano gli interrupt, perché il processore non può rispondervi: non saprebbe più tornare al programma originale se un'interruzione viene interrotta ancora una volta, proprio perché nessuna istruzione CALL ha salvato il PC del programma originale, e quindi il processore non saprebbe più ripristinare il contesto originale.

### **Descrivere come possono essere gestite le comunicazioni tra processore e le periferiche.**

Le periferiche comunicano con il resto del calcolatore grazie a delle schede opportune (interfacce) che scambiano dati su un BUS di I/O. L’interfaccia di elaborazione è costituita da una parte che si interfaccia con il calcolatore (riceve ed invia comandi e dati) e una parte che si interfaccia con le periferiche. Quest’ultima parte è connessa ad un opportuno cavo che termina sull’elettronica di controllo della periferica.

Questa elettronica di controllo è composta da una parte di gestione di connessione e una parte di controllo della periferica vera e propria (che si occupa della movimentazione di componenti della periferica (mantenimento della rotazione del disco e posizionamento delle testine di lettura e scrittura).

Per gestire tale canale all’interno dell'unità centrale esistono dispositivi di controllo appositi che costituiscono l’interfaccia della periferica. Questa contiene un insieme di registri che permettono il controllo con la periferica e il ritorno dei risultati delle operazioni svolte dalla periferica verso il calcolatore. In particolare abbiamo: un registro di stato, un registro comandi ed un registro dati.

L'unità centrale chiede all’interfaccia di verificare lo stato di comunicazione con la periferica ottenendo il contenuto del registro di stato: se la connessione è ok allora la periferica è libera e il calcolatore potrà quindi proseguire con l’esecuzione di un comando di I/O deponendo all’interno del registro dati l’ordine da fare pervenire alla periferica.

L’unità centrale scriverà nel registro comando l’ordine da inviare e i dati richiesti alla periferica. L’elettronica dell’interfaccia comprenderà che è necessario inviare alla periferica i dati richiesti nel registro comando. Ciò viene effettuato trasferendo il contenuto del registro dati a bordo dell’interfaccia di I/O del calcolatore che controlla la connessione con la periferica, la quale trasferirà l’ordine della richiesta presente nel registro dati all’interno della periferica stessa.

Allora i dati verranno interpretati, verrà eseguito il comando e verrà ritornato eventualmente il risultato all’interno del registro dati, aggiornando il sistema dell’interfaccia della periferica in modo tale che il processore venga informato del risultato della richiesta effettuata.

Il trasferimento dei dati tra CPU e interfaccia può avvenire in diversi modi:

* per singola parola di memoria: modo appena visto attraverso il registro dati. Non funziona in maniera efficace ed efficiente quando si devono trasferire tanti dati.
* a blocchi: usando il Direct Memory Access (DMA), consentendo alla CPU di essere più efficiente nel trasferimento di dati, permettendo il trasferimento di un byte dopo l’altro fino al termine del trasferimento.

# 

# M2 - ARCHITETTURA DEI SISTEMI OPERATIVI

## UD1 - STRUTTURA DEI SISTEMI OPERATIVI

### **Un sistema operativo è caratterizzato principalmente da due finalità, quali?**

* **Astrazione dei componenti della macchina**: l’innalzamento del livello di astrazione dei componenti permette di semplificare l’accesso alle risorse e mostrare ai programmi e agli utenti un comportamento più semplice delle varie risorse, nascondendo dietro l’astrazione i dettagli di gestione delle periferiche
* **Virtualizzazione delle risorse**: l'obiettivo è creare un immagine del sistema di elaborazione dedicata a ciascun programma in esecuzione anche se in realtà i vari programmi vengono eseguiti un po uno un po l’altro. Questo ha come obiettivo quello di fornire la possibilità ai programmi di non considerare la presenza di altri programmi e ha anche come obiettivo quello di semplificare la programmazione dei software. Ogni programma vede quindi una macchina “tutta per sé” quando in realtà le risorse della macchina sono condivise.

### **Quali servizi deve fornire un sistema operativo?**

Un sistema operativo deve essere in grado di fornire servizi atti alla gestione del processore, della memoria centrale, periferiche I/O e file system.

* Per la gestione del processore è necessario avere ad esempio delle funzioni per la gestione dei processi: creazione e terminazione dei processi, sospensione e riattivazione dei processi, schedulazione dei processi, sincronizzazione tra processi, gestione di situazioni di stallo (deadlock), comunicazione tra processi.
* Per la gestione della memoria centrale: queste funzioni costituiscono il supporto per la multiprogrammazione, devono provvedere alla allocazione e deallocazione della memoria ai processi, devono provvedere al caricamento e allo scaricamento di processi e di loro porzioni in memoria centrale, garantendo la protezione della memoria.
* Per la gestione delle periferiche si ha bisogno di fornire una omogeneità di interazione: deve consentire la configurazione e inizializzazione, gestione ottimizzata dei dispositivi di ingresso, uscita, memorizzazione di massa e rete informatica, protezione delle periferiche, bufferizzazione e caching.
* Per la gestione del file system. il file system deve dare all’utente la possibilità di creare e cancellare file e cartelle, leggere e scrivere, copiare, ricercare, proteggere, gestirne l’accounting e permettere il ripristino.
* Per la gestione dell’interfaccia grafica: Interfacciarsi con il SO vuol dire fornire quelle funzioni che permettono di dare comandi e ricevere i risultati. Devono quindi essere presenti le funzioni: interprete comandi a livello utente e a livello applicativo (chiamate di sistema), librerie di sistema (modo con il quale i comandi possono interagire con il resto del sistema), gestione delle autenticazioni e gestione degli errori e dei malfunzionamenti.

### **Per ciascuno dei sistemi operativi di seguito riportati si fornisca una breve descrizione delle proprietà principali:**

* 1. **Per computer palmare**

Un computer palmare è un elaboratore di piccole dimensioni, solitamente dotato di processori poco potenti (ma a basso consumo d'energia) e di quantità di memoria RAM limitate. Il sistema operativo, in questo caso, dovrà avere una footprint relativamente piccola, dovrà fornire la gestione di un file system (solitamente salvati su memorie di tipo flash) e dovrà fornire un'interfaccia utente appropriata (grafica, con gestione del touch screen). La schedulazione avverrà su un numero di processi, nella maggior parte dei casi, limitato. La schedulazione deve permettere all'utente di vedere un sistema "reattivo", per es usando quanti di tempo più brevi che in sistemi server e/o diversi algoritmi di schedulazione.

* 1. **Interattivo**

I sistemi operativi dei mainframe vennero ridotti in scala per soddisfare le esigenze di piccoli gruppi di utenti a costi contenuti, questi sistemi sono tipicamente dipartimentali e consentono una forte interazione con gli utenti e sono multiprogrammati, multi processati e dunque time-sharing. Ai sistemi interattivi sono succedute le workstation per portare il sistema “sul tavolo” del singolo utente.

* 1. **A lotti**

Sistemi operativi utilizzati soprattutto in computer mainframe dove inizialmente come input venivano prese delle schede perforate e l’output veniva restituito su una stampante, successivamente vennero adottati i nastri magnetici sia come input che come output. Questo tipo di sistema operativo a lotti (batch) venne così definito in quanto il programma veniva caricato, eseguito, e si passava poi al successivo.

* 1. **Di rete**

Questo tipo di sistemi operativi si basano essenzialmente su terminali adatti all’elaborazione basata sul web e vengono utilizzati al posto delle tradizionali workstation quando è richiesta una sicurezza maggiore o una manutenzione più semplice.

* 1. **Per architetture parallele**

Architettura con più processori strettamente connessi tramite bus PCI di sistema, le attività dei processori possono essere identiche quindi si ha un sistema multiprocessore simmetrico, viceversa, si possono avere unità di elaborazione specializzate nell’esecuzione di un particolare compito quindi si ha un sistema multiprocessore asimmetrico.

* 1. **In tempo reale**

Sistemi operativi orientati ad applicazioni in tempo reale, tipicamente utilizzati nel controllo di processi industriali, automazione industriale, domotica, ecc. Obiettivo di questi sistemi è quello di rispondere in tempi brevi agli eventi che si verificano nel mondo esterno. L’architettura hardware deve essere quindi in grado di scambiare segnali con il mondo esterno per poter supportare queste tipologie di interazione. Il sistema operativo deve adeguarsi in modo tale da garantire che gli eventi esterni siano sentiti e gestiti dal sistema nel tempo richiesto dall’applicazione.

* 1. **Time-sharing**

Sistemi che supportano molti utenti operanti contemporaneamente, ovvero, che permettono l’elaborazione contemporanea di flussi di attività condividendo tra i vari processi il tempo di esecuzione della CPU (time-sharing).

* 1. **Per cluster**

Architettura con più computer fortemente connessi, serve fondamentalmente a migliorare: capacità di elaborazione, economia di scala delle periferiche e l’affidabilità del sistema in caso di guasti.

* 1. **Per architetture distribuite**

Un sistema distribuito: un sistema distribuito è composta da diversi calcolatori collegati tra di loro tramite una rete di qualche tipo. Per la natura di questi sistemi, le informazioni ed i programmi in esecuzione sono distribuiti sui diversi calcolatori presenti. In questo caso il SO dovrà gestire, oltre ai normali servizi, la comunicazione tra i diversi calcolatori. In particolare dovrà gestire l'esecuzione remota di processi e l'accesso remoto a file.

### **Che differenza c'è tra multiprogrammazione e multitasking?**

L'obiettivo della multiprogrammazione è di avere più processi caricati in memoria centrale in modo da consentirne l'esecuzione contemporanea dal punto di vista macroscopico degli utenti, al fine di massimizzare l'utilizzo della CPU.

E' un'estensione logica logica della multiprogrammazione. L'obiettivo della gestione a condivisione del tempo (time sharing) è quello di alternare l'uso della CPU tra i processi così frequentemente che gli utenti possano effettivamente interagire con i propri processi mentre sono in esecuzione, come se evolvessero tutti in parallelo. Il multitasking richiede un sistema interattivo in modo tale che ci sia una comunicazione diretta tra utente e sistema.

[La multiprogrammazione consiste nel caricamento di più programmi contemporaneamente nella memoria centrale.  
Il multitasking, o multiprocessualità, gestisce al meglio la multiprogrammazione eseguendo una parte di processo alla volta per massimizzare l’efficienza di calcolo eliminando il problema dei tempi morti, che rallentano notevolmente l’operatività di un sistema, a livello tale da dare l’illusione agli utenti che i processi da loro utilizzati progrediscano in contemporanea e permettendo così una maggiore reattività ai comandi degli utenti e l’ottimizzazione della gestione delle risorse del sistema.]

### **In accordo con le diverse definizioni che si possono avere di sistema operativo, si indichi se sia possibile o meno considerare come parti dello stesso anche le seguenti applicazioni: server di posta elettronica e di applicazioni web. Giustificare in maniera chiara sia le tesi a favore che contro le proprie risposte.**

Generalmente, le applicazioni come server di posta elettronica e server di applicazioni web non sono da considerarsi elementi costituenti di un sistema operativo in quanto non fanno parte dei servizi elementari offerti da un sistema operativo per l’astrazione dell’hardware. Tutt'al più sono da considerarsi come programmi eseguiti nello spazio utente, infatti, eseguire questi programmi nello spazio utente porterebbe con sé diversi vantaggi, ad esempio: se uno dei due servizi viene compromesso, il resto del sistema operativo rimane intatto, il che garantisce una maggiore sicurezza e affidabilità del sistema, inoltre, se il kernel non supporta moduli caricabili dinamicamente, in caso ci sia necessità di aggiornare gli applicativi, si deve poi riavviare il sistema per applicare gli aggiornamenti.

In linea teorica si potrebbero comunque considerare i due servizi come parte di un sistema operativo in determinate strutture: si può pensare alla struttura modulare o monolitica dove i due servizi indicati possono essere all’interno del kernel del sistema operativo offrendo così ulteriori funzionalità oppure caricate come moduli aggiuntivi.

### **Architetture dei sistemi operativi.**

**Sistema monolitico**

Un sistema contenitore in cui venivano inserite funzioni senza una specifica strutturazione. Le funzioni di livello più basso (più semplici) possono richiamare le funzioni di livello più alto, portando confusioni e una manutenzione più difficoltosa.

Il vantaggio principale di quest’approccio è che il sistema è compatto, veloce ed efficiente, quindi è un approccio adatto a sistemi semplici.

**Sistema con struttura gerarchica**

Un sistema con struttura gerarchica, ovvero una organizzazione delle funzioni a livelli dipendenti tra loro. In particolare si sono creati vari livelli funzionali dal più vicino alla struttura hardware al più astratto, più vicino cioè all'interfaccia utente. Si è poi stabilito che ogni livello potesse comunicare esclusivamente con livelli di strato inferiore, anche se non necessariamente con quello immediatamente successivo.

Questa nuova organizzazione ha reso più semplice la manutenzione rispetto ai precedenti sistemi monolitici, ma racchiude in sé ancora dei limiti derivanti in particolare dalla non chiara suddivisione in ruoli delle singole funzioni. Inoltre anche la progettazione risulta un compito difficile, in quanto è necessario stabilire a priori a quale livello deve appartenere la funzione da implementare.

**Sistema stratificato**

Questi introducono una chiara separazione modulare delle funzioni svolte da ciascun componente del sistema operativo. Si parte quindi da un modulo di base per la gestione del microprocessore, su cui poggia il modulo per la gestione della memoria che a sua volta funge da base per il modulo di gestione delle periferiche e così via fino al modulo di interfaccia utente.

In questa maniera le varie componenti sono gerarchicamente suddivise in livelli funzionali, che permettono una più semplice manutenzione del sistema, penalizzando però l'efficienza dell'intero sistema, perdendo tempo nel passaggio e nelle comunicazioni tra un livello e l'altro.

**Sistema a microkernel**

Separazione dei meccanismi (serie di operazione di gestione di accesso alle singole risorse) dalle politiche (definizione astratto del diritto/ordine dell’uso della risorsa). Permettendo anche di aggiornare e modificare il sistema senza mai fermarlo.

Per meccanismi si intendono tutte quelle operazioni, fisse, raramente modificabili, di accesso alle risorse, mentre le politiche rappresentano quel genere di definizioni più astratte di diritto, piuttosto che priorità o ordine di uso di una risorsa. Nasce così il concetto di microkernel ovvero a tutti gli effetti un kernel ma scremato di quelle componenti non strettamente necessarie alla gestione dei meccanismi. Tutto il resto, ovvero tutto ciò che è politica, viene organizzato in processi al di sopra del microkernel.

Con questa struttura i programmi utente in esecuzione non comunicano direttamente con i servizi messi a disposizione dal sistema operativo, ma comunicano indirettamente mediante lo scambio di messaggi con il microkernel. Questo è il limite principale del sistema, perché ne riduce le prestazioni. In compenso si ha un ottima modificabilità e la possibilità di estendere il sistema senza modificarne il kernel, inoltre vi è una maggiore affidabilità in quanto la maggior parte dei servizi funziona in modalità utente.

**Sistema a moduli funzionali**

Un'ultima evoluzione dell'architettura dei sistemi operativi, nell'ottica di una sempre più semplice ed efficiente espansione e manutenzione globale del sistema fa uso delle moderne tecniche di programmazione ad oggetti. Ciò permette di costruire kernel modulari a cui è possibile aggiungere o togliere dinamicamente delle componenti a seconda delle esigenze. Si torna quindi ad avere un kernel limitato alle funzioni di base intorno al quale vengono agganciati moduli diversi in base alle esigenze del momento.

Questo approccio ha diversi punti in comune con le soluzioni a strati (le interfacce di interazione con i diversi moduli) ed a microkernel (un kernel con funzioni basilari), è però più efficiente in quanto ogni funzione non è vincolata a richiamarne solo di appartenenti a livelli inferiori ed ognuna di esse può comunicare con le altre direttamente, senza scambiare messaggi con il kernel.

**Sistema a macchine virtuali**

Per astrarre ulteriormente questo concetto sono state introdotte le macchine virtuali nelle quali si ha una costruzione gerarchica di macchine astratte caratterizzate dall’hardware, al di sopra si ha un kernel della VM che provvede a replicare l’hardware replicandolo esattamente per ogni insieme di processi che si vogliono eseguire sulla macchina virtuale, dove si potrà installare un sistema operativo. Su ciascuno di questi ambienti diversi si avranno dei processi in esecuzione.

Ciascuno vedrà il proprio SO senza sapere che in realtà ci sono altri ambienti operativi disponibili sul sistema. La virtualizzazione permette di usare contemporaneamente diversi operativi.

Questa tecnica non mette a disposizione nessuna nuova funzionalità, ma fornisce un'interfaccia identica all'hardware sottostante gestita dal kernel di macchina virtuale che è in grado di generare più macchine virtuali su cui installare sistemi operativi diversi. Ottenendo così la convivenza di sistemi operativi eterogenei. Il limite più grande lo si ha nel considerevole calo di prestazioni.

# 

# M3 - GESTIONE DEL PROCESSORE

## UD1 - Processi

### **Che differenza c'è tra programma e processo?**

I primi sono entità passive, una sequenza di istruzioni contenute in un file sorgente salvato su disco. I processi sono invece entità attive, con un program counter che specifica l'istruzione successiva da eseguire, un gruppo di istruzioni in uso e una particolare istanza dei dati su cui era stato mandato in esecuzione il programma.

E' per questo che posso benissimo avere due processi associati allo stesso programma, verranno comunque considerati come istanze di esecuzione distinte dello stesso codice. Ad esempio, posso aprire due finestre di Firefox e gestirle separatamente: il loro sorgente è uguale, la sezione dati è differente.

### **Cos'è lo stato di un processo?**

Un processo durante l’esecuzione è soggetto a cambiamenti di stato, definito in parte dall’attività corrente del processo stesso. Un processo può trovarsi in uno dei seguenti stati:

* Nuovo: si crea il processo
* Esecuzione (running): le istruzioni vengono eseguite
* Attesa (waiting): Il processo attende che si verifichi qualche evento come il completamento di un’operazione di I/O o la ricezione di un segnale
* Pronto (ready): Il processo attende di essere assegnato a un unità di elaborazione
* Terminato: Il processo ha terminato l’esecuzione

In ciascuna unità di elaborazione può essere in esecuzione un solo processo per volta, sebbene molti processi possano essere pronti o nello stato di attesa.

Le informazioni riguardanti lo stato di un processo sono memorizzate nel PCB insieme alle informazioni sullo scheduling della CPU.

### **Il diagramma degli stati di un processo descrive l'uso delle risorse del sistema di elaborazione da parte di un processo in esecuzione?**

Si, in realtà rappresenta l’insieme degli stati del processo e le transizioni tra stati. Tutto questo tramite un Grafo Orientato con i nodi che rappresentano gli stati e gli archi le transizioni. Durante la creazione (nuovo) vengono configurate tutte le informazioni relative alla sua gestione, viene riservato lo stato in memoria centrale, si dichiarano le risorse di cui ha bisogno, vengono fatte delle operazioni di memorizzazione delle richieste di uso di risorse dal programma per poi effettuare addebiti o altri tipi di statistiche.

Infine il S.O. pone il processo in pronto entrando in competizione per l’uso della CPU. Sarà poi il dispatching il meccanismo (del S.O.) per la scelta del processo attraverso dei criteri.

Una volta entrato in esecuzione il processo sarà abilitato all'uso del processore. Ciò non toglie che potrebbe mettersi in attesa per un evento o per un'operazione di I\O cosi come potrebbe essere interrotto forzatamente. Allo stesso modo può, invece, riuscire a terminare il proprio compito. Completato l'accounting viene poi rimosso dal sistema. In caso di terminazione per un errore potrà anche attivare il post mortem debugging.

### **Gli sviluppatori di un nuovo sistema operativo hanno deciso, per fare meno fatica, di non supportare la programmazione concorrente. Quali problemi li hanno spinti a prendere questa decisione? Citare i tre principali.**

* Overhead nel passaggio tra un processo ad un altro (cambio di contesto)
* Sincronizzazione degli accessi alle risorse tra processi diversi (implementazione di uno scheduler)
* Bilanciamento dei processi (eseguire pochi processi I/O-bound non sfrutterebbe al meglio il processore, dandogli poco e niente da computare; eseguire troppi CPU-bound invece, pur assicurando un uso intensivo della CPU, rallenterà drasticamente il sistema, il processo monopolizza il processore e tutti gli altri non potranno evolvere)

### **Che operazioni deve eseguire il kernel per un cambio di contesto fra processi?**

Supponendo di avere in esecuzione sul processore il processo P0, quando questo viene sospeso si provvede a salvare lo stato di esecuzione sul PCB0, quindi si carica lo stato salvato sul PCB1 e lo si manda in esecuzione, cambiando contesto e passando all'esecuzione del processo P1.

Mentre è in esecuzione P1, il precedente processo P0 si trova in una situazione di attesa. Appena P1 viene sospeso, si provvede a salvare lo stato di esecuzione nel PCB1, si carica lo stato precedentemente salvato sul PCB0 e si ritorna all'esecuzione del processo P0.

Nel momento in cui si ripristina il contesto di esecuzione di P0 si ricrea esattamente la stessa situazione di esecuzione che era presenta al momento in cui il processo era stato sospeso.

Quindi dal punto di vista dell'esecuzione della computazione di P0 è come se avvenisse un "salto" tra il momento della sospensione e quello della riattivazione.

Il Kernel per effettuare un cambio di contesto tra processi deve svolgere le seguenti operazioni:

1. Sospendere l'esecuzione del processo P0 attualmente in esecuzione utilizzando il meccanismo di pre-emption
2. Salvare sullo Stack del processo P0 il contenuto dei registri del processore
3. Salvare sullo Stack del processo P0 il contenuto del Program Counter
4. Salvare sul PCB di P0 il valore dello Stack Pointer
5. Inserire nella 'coda dei processi pronti' il PCB del processo P0
6. Prelevare dalla 'coda dei processi pronti' il PCB del nuovo processo che verrà eseguito (P1)
7. Leggere il valore dello Stack Pointer contenuto nel PCB di P1
8. Ripristinare i valori dei registri letti dallo Stack associato al processo P1
9. Ripristinare il valore del Program Counter letto dallo Stack associato al processo P1

## UD2 - Thread

### **Descrivere due esempi di applicazione in cui la realizzazione multithread offra prestazioni migliori di una soluzione basata su un unico thread.**

I vantaggi della programmazione multithread si possono classificare in quattro categorie principali:

1. **Tempi di risposta** : rendere multithread un’applicazione interattiva può permettere a un programma di continuare la sua esecuzione anche se una parte di esso è bloccata sta eseguendo un’operazione particolarmente lunga, riducendo il tempo di risposta all’utente
2. **Condivisione delle risorse** : I processi possono condividere risorse soltanto attraverso tecniche come la memoria condivisa e lo scambio di messaggi. Tuttavia i thread condividono per default la memoria e le risorse del processo al quale appartengono.
3. **Economia** : Assegnare memoria e risorse per la creazione di nuovi processi è costoso; poichè i thread condividono le risorse del processo cui appartengono, è molto più conveniente creare thread e gestirne i cambi di contesto.
4. **Scalabilità** : Nelle architetture multiprocessore, i thread si possono eseguire in parallelo su distinti core di elaborazione. Invece un processo a singolo thread può funzionare su un solo processore, indipendentemente da quanti ve ne siano a disposizione.

### **Quali sono le due differenze principali fra i thread a livello utente e quelli a livello kernel? Descrivere le circostanze in cui una soluzione è migliore dell'altra.**

I thread a livello kernel sono gestiti direttamente dal kernel e sono quelli che vengono schedulati dallo stesso.

I thread a livello utente sono gestiti tramite librerie, anche senza supporto da parte dal kernel, che potrebbe addirittura non supportare nessun tipo di multithreading. Questi thread necessitano di essere mappati su uno o più processi/thread a livello kernel. Saranno questi ultimi ad essere effettivamente schedulati dal kernel; le librerie a livello utente si occuperanno poi di gestire i kernel a livello utente.

Quando un thread a livello kernel viene schedulato, gli viene effettivamente assegnata la CPU. Quando invece un thread a livello utente viene schedulato dalla libreria, non gli viene effettivamente assegnata la CPU. Quest'ultima gli verrà effettivamente assegnata quando il corrispondente thread a livello kernel andrà in esecuzione.

Esistono 3 diverse modalità per mappare thread a livello utente su thread a livello kernel: uno a uno, molti a uno e molti a molti. Nella modalità uno a uno ciascun thread a livello utente ha associato un thread a livello kernel. Nella modalità molti ad uno, un thread a livello kernel ha associato più thread a livello utente. Nella modalità molti a molti molti thread a livello utente vengono associati ad un numero più piccolo o equivalente di kernel thread.

I thread a livello utente offrono il beneficio di avere migliori prestazioni, poiché non sono necessarie chiamate di sistema per passare da un thread all'altro; d'altra parte, essendo la granularità maggiore, quando un thread a livello utente si blocca, potrebbe accadere che tutto il processo (o tutto l'LWP) si blocchi.

Diversi thread a livello kernel possono facilmente essere assegnati a processori diversi, nel caso di sistemi multiprocessore; i thread a livello utente, invece, non possono godere di questo vantaggio.

L'altro grande vantaggio dei thread a livello kernel consiste nella possibilità di tenere traccia con maggiore precisione dello stato di ogni thread, cosa che invece non è sempre possibile garantire con i thread a livello utente.

Un caso in cui i thread a livello utente vincono nettamente sui thread a livello kernel è quando sono necessari moltissimi thread: poiché ogni thread usa della memoria, spesso il numero massimo di thread a livello kernel è limitato; mentre i thread a livello utente, utilizzando memoria del processo, hanno limiti molto più alti.

I thread a livello kernel sono particolarmente vantaggiosi nei casi in cui c'è elevata contention su risorse e su primitive di sincronizzazione (mutex ecc): in questo caso, dovendo comunque effettuare molte chiamate di sistema, il vantaggio principale dei thread in modalità utente scompare e la differenza di velocità sparisce.

### **Descrivere le azioni che il kernel esegue durante un cambio di contesto fra thread a livello kernel.**

Le operazioni necessarie per il cambio di contesto di un thread a livello kernel sono analoghe a quelle per un processo, in particolare, si possono distinguere due casistiche: Se il cambio di contesto viene effettuato tra thread di uno stesso processo vi sarà un cambio di contesto semplificato in quanto per la natura dei thread i dati sono condivisi, mentre, nel caso in cui i thread appartengano a processi diversi vi sarà un cambio di contesto tra processi.

## UD3 - Schedulazione

### **Perché esistono tre schedulatori diversi? Come si chiamano e in cosa si differenziano? E' possibile eliminarne uno, due o tutti e tre?**

### **In questo esercizio si chiede di considerare una variante dell'algoritmo di schedulazione RR dove gli elementi della coda dei processi pronti sono puntatori ai PCB.**

* 1. *Quale sarebbe l'effetto di mettere due puntatori allo stesso processo nella coda dei processi pronti?*

L'effetto sarebbe una doppia esecuzione, una consecutiva all'altra, dello stesso processo. Quindi aumenterei il tempo di esecuzione di quest'ultimo. Dal momento che la coda contiene solo i puntatori, le successive iterazioni dello stesso processo dovrebbero fare riferimento allo stesso PCB, e quindi come il processo viene eseguito per 2 quanti di tempo invece di 1.

Se invece la coda contenesse i PCB veri e propri, il secondo PCB dello stesso processo non verrebbe aggiornato, con il risultato che in quel caso la stessa porzione di processo verrebbe eseguita due volte (con possibili risultati inconsistenti, se la prima esecuzione ha modificato i dati in memoria).

* 1. *Descrivere i due maggiori vantaggi e svantaggi di questo schema.*

Il primo vantaggio che mi salta in mente è che lo scheduler è più "snello" nel senso che non deve contenere il PCB per intero con tutte le sue info annesse. Un altro vantaggio credo sia l'assenza del context switch siccome il puntatore punta sempre allo stesso PCB.

Lo svantaggio invece può esserci nel momento in cui il processo termina prima del termine del primo quanto di tempo. In questo caso bisogna gestire il secondo puntatore siccome il processo è terminato e probabilmente non è più necessaria una sua esecuzione.

* 1. *Quale modifica dell'algoritmo RR di base permette di ottenere lo stesso effetto senza duplicare i puntatori?*

In genere il RR viene modificato andando a variare la durata del quanto di tempo quindi presumo che si possano ottenere gli stessi risultati di cui sopra andando ad aumentare la durata del quanto di tempo relativo a quel processo.

In aggiunta alla modifica che dice lei, si può anche utilizzare un algoritmo di schedulazione a code multiple, e impostare quanti di tempo differenti per ogni categoria. Invece di duplicare il puntatore al processo, lo si inserisce in una coda che ha un quanto di tempo più alto.

### **Descrivere i vantaggi che si ottengono nel fissare differenti dimensioni del quanto di tempo su livelli diversi di un sistema a code con più livelli.**

### **Considerare nel seguito un algoritmo di schedulazione preemptive con priorità dinamiche, dove numeri di priorità più alti implicano priorità più alte. La priorità di un processo che aspetta la CPU (nella coda dei processi pronti, ma non in esecuzione) cambia di una quantità a, mentre la priorità di un processo in esecuzione cambia di una quantità b. Tutti i processi hanno inizialmente priorità 0 quando entrano nella coda dei processi pronti. Considerando che i valori dei parametri a e b possono essere modificati per dare algoritmi di schedulazione differenti, descrivere:**

* 1. l'algoritmo che risulta da β > α > 0;
  2. l'algoritmo che risulta da α < β < 0.

### **Si considerino i seguenti algoritmi di schedulazione:**

* 1. FCFS;
  2. RR;
  3. code multilivello con feedback.

Spiegare le differenze nel trattamento dei processi brevi.

## UD4 - COMUNICAZIONE TRA PROCESSI

### **Una comunicazione può essere associata a più di due processi?**

### **Come avvengono le comunicazioni tra processi con memoria condivisa?**

### **Cosa sono e come avvengono le comunicazioni tra processi attraverso buffer di sistema operativo?**

### **Come avvengono le comunicazioni mediante mailbox tra due processi?**

### **Che differenza c'è tra le comunicazioni tra processi e le comunicazioni tra thread?**

## UD5 - SINCRONIZZAZIONE DEI PROCESSI

### **Spiegare il motivo per cui l'attesa di un processo non viene realizzata tramite un'istruzione condizionale.**

### **La soluzione al problema della cena dei filosofi non previene la starvation di un filosofo. Per esempio i due filosofi filosofo1 e filosofo3, potrebbero alternare il mangiare ed il pensare impedendo a filosofo2 di mangiare. Come può essere risolto questo problema utilizzando la sincronizzazione?**

### **Il problema del barbiere dormiente. Un negozio da barbiere è composto da una sala con una poltrona da barbiere e da una sala d'attesa con n sedie. Il barbiere va a dormire quando non ci sono clienti da servire. Un cliente esce dal barbiere se entra e tutte le sedie sono occupate, altrimenti si accomoda in sala d'attesa se il barbiere è occupato ma ci sono sedie libere. Se il barbiere sta dormendo, il cliente lo sveglia. Descrivere come potrebbe essere risolto il problema del barbiere dormiente, sincronizzando barbiere e clienti.**

### **Descrivere il concetto dell'atomicità delle transazioni.**

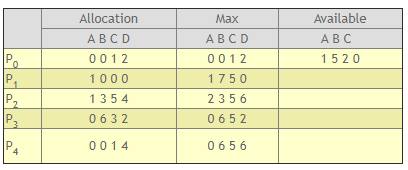
## UD6 - DEADLOCK

### **In un sistema di elaborazione reale, sia le risorse disponibili sia le richieste dei processi per le risorse non risultano essere consistenti per lunghi periodi di tempo. Se l'algoritmo del banchiere evita il deadlock, quali dei seguenti cambiamenti possono essere apportati in modo sicuro, senza introdurre la possibilità di deadlock? In quali circostanze tali cambiamenti possono avvenire?**

* 1. Aumentare il numero di risorse disponibili;
  2. Diminuire il numero di risorse disponibili;
  3. Aumentare il numero di risorse disponibili per un processo;
  4. Diminuire il numero di risorse disponibili per un processo;
  5. Aumentare il numero di processi;
  6. Diminuire il numero di processi.

### **Un computer fa girare 5.000 job al mese senza alcun meccanismo per prevenire ed evitare i deadlock. I deadlock si presentano circa due volte al mese e l'operatore deve terminare e ripetere circa 10 job a causa di ogni deadlock. Ogni job vale circa $2 (tempo della CPU) e i job terminati sono svolti circa per metà prima di venire abortiti. Un algoritmo per evitare il deadlock potrebbe essere installato nel sistema con un aumento nel tempo di esecuzione medio per esecuzione di circa il 10%. La macchina è attualmente in idle per il 30%; tutti i 5.000 job al mese potrebbero quindi essere ancora eseguiti, anche se il tempo di esecuzione aumenterebbe in media di circa 20%. Mostrare vantaggi e svantaggi dell'installazione di un algoritmo per evitare il deadlock nella situazione presentata.**

### **Si consideri il seguente sistema:**



Usando l'algoritmo del banchiere, rispondere alle seguenti domande:

* 1. Qual è il contenuto della matrice Need?
  2. Il sistema è in uno stato sicuro?
  3. Se processo P1 fa una richiesta per (0,4,2,0), tale richiesta può essere soddisfatta immediatamente?