SOCKET > 3.8.1

I Socket sono le estremità di un canale di com.
Sono formati dalla coppia IP: Porta che identifica l'elem. comunicante.

CLIENT SERVER

Sono usati nell'architettura client-server in cui il client è sender dei messaggi e il server rimane in attesa dei vari client.

TIPOLOGIE DI SOCKET

Possono essere:

- TCP: Focalizza la qualità e ricezione dei messaggi
- UDP: Connection-less, non fa check di ricezione
- Multicast Socket:
  Consente l'invio a diversi destinatari

La comunicazione con i socket viene considerata a basso livello perché gli interlocutori si scambiano byte non strutturati.

RPC > 3.8.2

Le Remote Procedure Calls sono un altro canale di comunicazione remoto.
Consistono nell'invocazione da parte del client di una procedura/funzione nel server.

OBBLIGHI DEL SERVER

Per funzionare, il server deve fornire informazioni aggiuntive come la firma del metodo.

OBBLIGHI DEL CLIENT

Il client deve serializzare i parametri verso il server (marshalling) per rendere l'informazione "digeribile" da entrambi.

Per l'implementazione si usano degli stub, dei proxy.

PIPE > ls | cat - 3.7.4

È un metodo che consente di mettere in connessione Producer e Consumer attraverso un Tubo che porta i dati al Consumer.

FUNZIONAMENTO GENERICO (su Linux)

Solitamente i comandi per "stampare a schermo" vanno a scrivere nel file descriptor dello schermo (stdout) Quindi la pipe, al posto di scrivere nello stdout, scriverà nello stdin del programma consumer.

Le pipe possono essere

- **Convenzionali o Un-named**

  Consentono una comunicazione uni-direzionale, che è riservata solo al processo che la crea (quindi anche ai suoi figli):

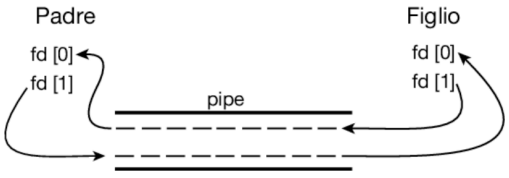


Figura 3.20 Descrittori di file per una pipe convenzionale.

Alla pipe si può accedere tramite il file descriptor:

0 -> In lettura

1 -> // scrittura

- **Named Pipes**

  Consentono una comunicazione bi-direzionale (Half duplex in linux e full Duplex in windows) ed è aperta a tutti i processi e anche a comunicazioni N:N.

  Il S.O. manterrà attiva la pipe come file anche quando i processi hanno finito.

PROGRAMMAZIONE MULTICORE > (2.2)

È un tipo di sviluppo che consente di sfruttare più core del sistema per efficientare un operazione.

Concorrenza vs Parallelismo

Sono concetti ben distinti:

- Concorrenza: Consente di eseguire più task facendoli progredire nell'esecuzione con l'interleaving dando l'illusione di parallelismo.
- Parallelismo: Consente di eseguire più task in parallelo.

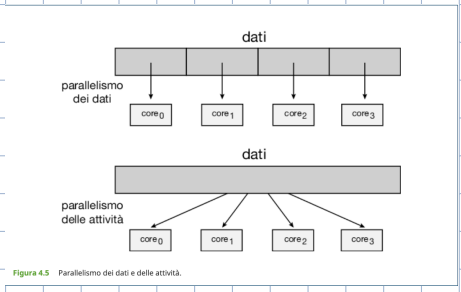


Figura 4.5 Parallelismo dei dati e delle attività.

Tipologie di Parallelismo (4.2.2)

Possono essere:

- Data Parallelism

  I dati in esame vengono divisi nell'elaborazione su più core. A sua volta si possono identificare come:

  - Map/Apply-to-all:

    Applico una trasformazione su tutti gli elementi

  - Reduce:

    Riduco i dati ad 1 elemento (es: Somma)

- Task Parallelism

  Prevede la distribuzione delle attività su più core. Si possono identificare come

  - Pipeline: Si assegna ad ogni task un piccolo compito (catena di montaggio)
  - Farm: Ogni attività viene considerata uguale e viene quindi suddivisa su più core (es: le request ad un server)

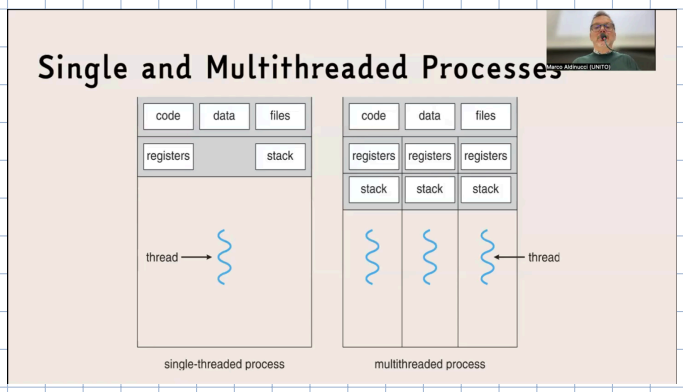
Passaggio a Multicore

I processi di default sono single core. Facendo il passaggio verranno

- Preservati: Text (codice), Data (Heap + var globali)

e Files (file aperti)

- Non Preservati: Stack e Registers



Quindi dove allocare le variabili è importante

LEGGE DI AMDAHL

È una legge usata per stimare il guadagno dovuto all'utilizzo del parallelismo:

$$\text{Guadagno} \leq \frac{1}{S + \frac{(1-S)}{N}}$$

Dove

S: % di codice sincrono

N: # di core

La retta è l'incremento ideale che è impossibile.