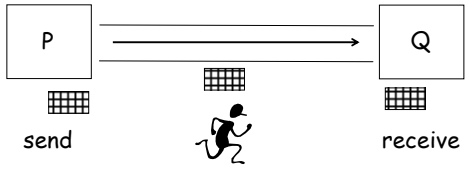
Cosa fare per comunicare ?!

Se i processi P e Q vogliono ***comunicare*** devono: stabilire un ***canale di comunicazione*** tra essi scambiare messaggi usando ***send*** e ***receive***



Ad un ***canale logico*** corrisponde un ***canale fisico*** che puo' essere o una cella di memoria condivisa oppure un bus hardware

Questioni implementative…

- Come si stabilisce una connessione?
- Puo' ***un canale*** essere associato a ***piu' di due processi***?
- Viceversa, ***quanti canali*** possono essere associati ***ad un'unica coppia*** di processi?
- Un canale e' uni- o bi-direzionale?

… le risposte dipendono dagli ***attributi principali*** di una comunicazione:

A. diretta/indiretta

B. buffering/no buffering

C. sincrona/asincrona

A. Communicazione diretta

Ognuno dei due processi deve conoscere il nome dell'altro processo con il quale vuole comunicare

contenuto

send (*P, message*) – manda ***message*** al processo ***P***
receive(*Q, buffer*) – riceve in ***buffer*** dal processo ***Q***

contenitore

Proprieta'

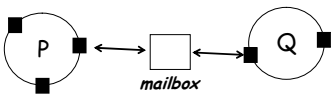
- La connessione e' stabilita in maniera automatica
- Un canale e' fissato tra due processi
- Fra due processi ci puo' essere un unico canale
- Il canale puo' essere bidirezionale

Comunicazione indiretta…

I messaggi sono mandati e ricevuti attraverso ***mailboxes*** (associati a ***porte***)

Ogni mailbox ha un ***identificatore unico***, noto ai processi che ne vogliono fare uso per comunicare

I processi possono comunicare solo se ***condividono*** un ***mailbox***



… operazioni , proprieta'…

Operazioni

Crea un mailbox
send e ***receive*** attraverso il mailbox
Distruggi il mailbox

Proprieta'

- La connessione e' stabilita solo se i processi condividono il mailbox
- Un mailbox puo' essere associato a piu' processi
- Fra due processi ci puo' essere piu' di un mailbox
- Il mailbox puo' funzionare in modo bidirezionale

… ed un problema

P1, P2, and *P3* condividono mailbox *A*
P1 manda messaggi; *P2* e *P3* ricevono

Chi ricevera' effettivamente i messaggi in arrivo?

Possibili soluzioni

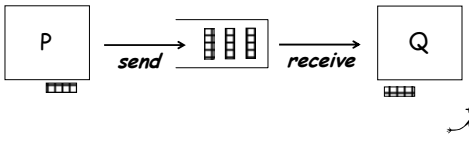
- Permettere che un mailbox sia associato a soli due processi
- Permettere che solo un processo alla volta possa eseguire una *receive*
- Delegare OS a selezionare arbitrariamente il ricevente ogni volta (al mandante verra' notificato dopo il nome di chi ha ricevuto)
- Multicasting e Broadcasting

B. Buffering / No buffering…

Un canale con buffering permette di associare una ***coda di messaggi*** ad esso

Questo significa che un processo che vuole inviare un messaggio lo puo' fare ***virtualmente*** se il canale e' gia' occupato

… e tre implementazioni possibili



Capacita' ***nulla*** – rendezvous (no buffering)
Capacita' ***finita*** – il sender aspetta solo se il canale e' pieno
Capacita' ***infinita*** – la comunicazione non e' mai ritardata

C. Sincrona / Asincrona…

Una operazione di **receive *e' sempre bloccante***
Una operazione di **send *puo' essere bloccante***

Comunicazione sincrona

Il processo che esegue la ***send*** si blocca attendendo che la corrispondente ***receive*** venga eseguita prima di proseguire

Un canale con ***buffer nullo*** puo' ospitare solo comunicazioni ***sincrone***

… e come rendere sincrona una comunicazione asincrona

Processo P	Processo Q	
send (*Q, message*)	**receive**(*P, buffer*)	*asincrona*
↓	↓	↓
send (*Q, message*)	**receive**(*P, buffer*)	*sincrona*
receive(*Q, buffer*)	**send** (*P, acknowledgment*)	

In alcuni OS l'operazione di ***send*** attende una ***reply*** entro un certo ***timeout***

Eccezioni da gestire per la comunicazione

- Un processo ricevente attende un messaggio da un processo che ha terminato
- Un processo invia un messaggio ad un processo che ha terminato (caso buffering e no buffering)
- Un messaggio inviato da un processo ad un altro viene perduto (timeout)
- Un messaggio inviato da un processo ad un altro viene danneggiato (il ricevente puo' notificarlo al mandante, se e' in grado di riconoscere l'errore)

Threads

Molto piu' esteso questo argomento sulla VI edizione !!!

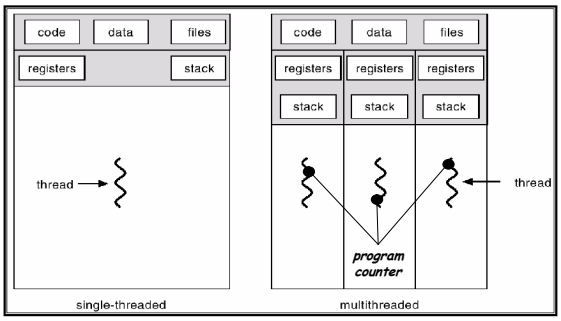
Una ***thread***, o ***processo leggero***, e' l'unita' di base dell'utilizzazione della CPU

Item ***propri*** di una thread ⟶
- *Stato*
- ***Program counter***
- *Insieme di registri*
- *Stack*

Item condivisi con le altre thread ad essa associate (a formare un ***task***) ⟶
- ***Sezione di codice***
- *Sezione dati*
- ***Risorse di OS***

Un ***processo pesante*** (tradizionale) e' uguale ad un task con un'unica thread

Multiple threads all'interno di un task



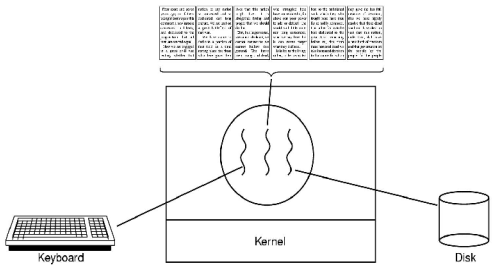
Alcune proprieta' delle thread…

- ***Una*** sola thread ***alla volta in esecuzione***
- Una thread puo' ***leggere*** o ***scrivere nello spazio*** di qualunque altra thread dello stesso task
- In caso di ***bloccaggio*** di una thread, ***un'altra*** dello stesso task ***puo' continuare*** la sua esecuzione, e quindi il processo non e' bloccato
- Il ***context-switching*** a livello di thread e' piu' rapido in quanto non coinvolge l'uso della memoria

… ed alcune considerazioni

- La cooperazione di multiple thread conferisce un piu' ***alto throughput*** e migliori performance
- Applicazioni che devono ***condividere risorse*** si giovano di thread, ex. un task che deve fornire dati a piu' macchine remote in un file system di rete
- Esse permettono di ***introdurre parallelismo*** nella esecuzione di task che eseguono system call bloccanti

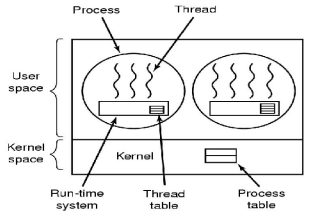
Per esempio: un text editor



Threads: gestione a livello utente

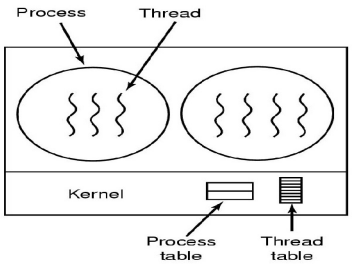
Un insieme di funzioni di libreria per gestire le thread

Il kernel allora vede ogni task come un singolo processo, le thread non sono unita' visibili da esso



- Passaggio veloce da una thread all'altra
- Bloccata una thread bloccato l'intero task
- Sbilanciamento nello scheduling

Threads: gestione a livello kernel



Un compito in piu' per OS :
gestire le threads
(scheduling, creazione, etc.)