Progetto I/O a 8 bit su Raspberry

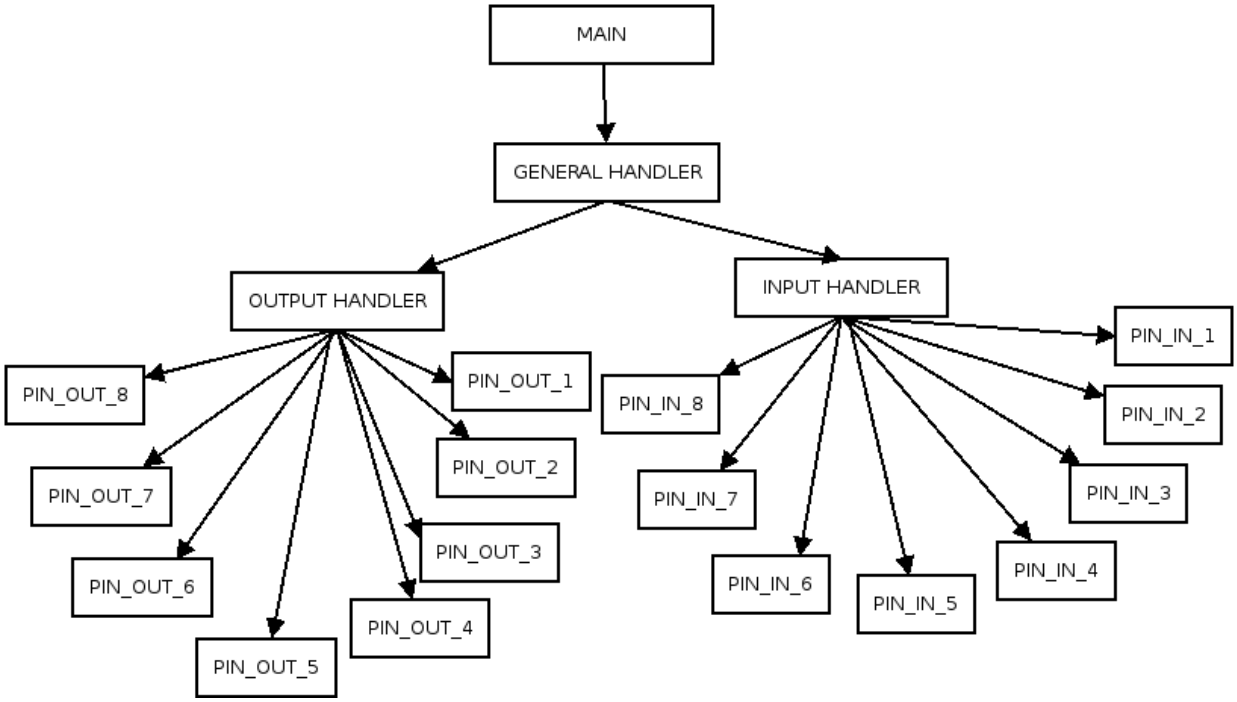
Nel progetto I/O a 8 bit su Raspberry è necessario leggere 8 valori in input (interpretati come singoli bit 1/0), eseguire delle operazioni su di essi e restituire altrettanti valori in output.

Per fare questo abbiamo creato 20 processi. Il processo main forka il processo general handler, il quale esegue il vero e proprio programma. Questo va a forkare altri due processi, i gestori di input e di output che sono incaricati di ricevere e inviare segnali ai pin del Raspberry.

L'input handler, dal momento che deve leggere i segnali di input (interpretati come dei bit) di 8 diversi pin, va a forkare 8 processi figli che serviranno a gestire singolarmente questi segnali.
Ad ognuno di questi 8 processi viene passato l'identificativo numerico del pin di input di cui deve andare a leggere il valore. Questi identificativi (insieme a quelli dei pin di output) vengono presi da un file di configurazione esterno.

Una volta ottenuti questi bit, il gestore generale di input va a salvarli in un array di 8 interi (in modo da salvare lo stato attuale). Questo array viene poi passato al general handler che andrà ad implementare la logica per trasformare questi 8 valori iniziali in un array risultante, secondo quanto stabilito dalle specifiche del progetto.

L'array modificato viene poi passato al gestore di output, il quale è incaricato di inviare i singoli valori dei bit ai rispettivi 8 pin di output. Per fare questo anch'esso va a forkare 8 processi e ognuno di questi andrà a scrivere il valore passato sul pin di output, così da accendere/spegnere il led a lui associato sulla breadboard (e avere un feedback esplicito di quanto si sta facendo).



Per implementare la comunicazione tra processi abbiamo usato le message queues. Una di queste è stata usata per far comunicare tra loro i processi di input, che gestiscono i singoli pin fisici, col processo di general input handler. Il corpo del messaggio è composto dal valore del pin (un bit 1/0); inoltre ogni messaggio ha un identificativo uguale all'identificativo numerico del pin, in modo da rendere possibile la distinzione tra messaggi.
Infatti, una volta che l'input handler riceve un messaggio, deve sapere a quale pin è relativo per scrivere il valore nella giusta posizione dell'array.

La seconda message queue implementata è stata usata per gestire il “passaggio” degli array di input e dell’array di output tra l’input handler, il general handler e l’output handler. Per distinguere i messaggi sono stati usati due valori: se il tipo del messaggio era 1 allora l’argomento del messaggio era l’input array inviato dall’input handler al general handler, mentre col valore 2 veniva identificato l’array di ouput inviato dal general handler all’output handler.
In questo modo, avendo due tipi diversi, non si rischia che due processi vadano a leggere qualcosa che non spetti a loro (in questo caso il general handler potrà leggere solo messaggi di tipo 1 mentre l’output handler solo quelli di tipo 2).

Infine la terza coda è quella usata tra il general output handler e gli 8 singoli processi dei pin di output. Come con la prima coda il messaggio trasportato è il valore da scrivere nel pin (bit 1/0) e l’identificativo è il riferimento numerico al pin. A differenza della prima coda (in cui l’input handler legge tutti i valori dei messaggi e poi li posizione nella giusta posizione dell’array) i processi in ricezione vanno solo a leggere i messaggi con tipo uguale al pin in cui devono andare a scrivere.