dueThread.c - poiché il main dopo avere creato i due figli, non aspetta i figli e fa una exit possiamo avere varie situazioni in output!

```
osd@lica04:~/Pthread/primeProve$ dueThread
Sto per creare il thread 0-esimo
Sto per creare il thread 1-esimo
osd@lica04:~/Pthread/primeProve$ dueThread
Sto per creare il thread 0-esimo
Sto per creare il thread 1-esimo
Thread0 partito: Hello World!
osd@lica04:~/Pthread/primeProve$ dueThread
Sto per creare il thread 0-esimo
Sto per creare il thread 1-esimo
osd@lica04:~/Pthread/primeProve$ dueThread
Sto per creare il thread 0-esimo
Sto per creare il thread 1-esimo
```

dueThread-conReturn.c – idem come sopra, dato che poiché il main dopo avere creato i due figli, non aspetta i figli e fa un return possiamo un output non corretto!

```
osd@lica04:~/Pthread/primeProve$ dueThread-conReturn
Sto per creare il thread 0-esimo
Sto per creare il thread 1-esimo
Thread0 partito: Hello World!
```

dueThread-conSleep.c –il main dopo avere creato i due figli, non aspetta i figli ma prima di eseguire la exit fa una sleep (di 10 secondi) e quindi otteniamo output corretto!

```
osd@lica04:~/Pthread/primeProve$ dueThread-conSleep
Sto per creare il thread 0-esimo
Sto per creare il thread 1-esimo
Thread0 partito: Hello World!
Thread1 partito: Hello World!
```

dueThread-conScanf.c –un risultato analogo a prima lo possiamo ottenere se il main dopo avere creato i due figli, non aspetta i figli ma prima di eseguire la exit fa una scanf e quindi si riesce a ottenere l'output corretto andando ad inserire il numero atteso solo dopo le stampe dei figli!

```
osd@lica04:~/Pthread/primeProve$ dueThread-conScanf
Sto per creare il thread 0-esimo
Sto per creare il thread 1-esimo
Thread0 partito: Hello World!
Thread1 partito: Hello World!
10
Processo creatore ha letto 10
```

dueThread-conPexitNelMain.c –un risultato corretto lo possiamo avere se il main dopo avere creato i due figli, non aspetta i figli ma esegue (invece che exit o return) una pthread_exit.

```
osd@lica04:~/Pthread/primeProve$ dueThread-conPexitNelMain
Sto per creare il thread 0-esimo
Sto per creare il thread 1-esimo
Thread0 partito: Hello World!
```

```
Thread1 partito: Hello World!
osd@lica04:~/Pthread/primeProve$ dueThread-conPexitNelMain
Sto per creare il thread 0-esimo
Sto per creare il thread 1-esimo
Thread1 partito: Hello World!
Thread0 partito: Hello World!
osd@lica04:~/Pthread/primeProve$ dueThread-conPexitNelMain
Sto per creare il thread 0-esimo
Sto per creare il thread 1-esimo
Thread0 partito: Hello World!
Thread1 partito: Hello World!
```

dueThread-conJoin.c –un risultato corretto lo possiamo avere se il main dopo avere creato i due figli, li aspetta e solo dopo esegue una exit (o return).

```
osd@lica04:~/Pthread/primeProve$ dueThread-conJoin
Sto per creare il thread 0-esimo
Sto per creare il thread 1-esimo
Thread0 partito: Hello World!
Thread1 partito: Hello World!
```

dueThread-conJoinEPexitNeiThread.c –come prima, ma i figli tornano al padre un risultato (1000+id che è il loro numero d'ordine).

```
osd@lica04:~/Pthread/primeProve$ dueThread-conJoinEPexitNeiThread
Sto per creare il thread 0-esimo
Sto per creare il thread 1-esimo
Thread0 partito: Hello World!
Thread1 partito: Hello World!
Pthread 0-esimo restituisce 1000
Pthread 1-esimo restituisce 1001
```

dueThread-conJoinEPexitNeiThreadESelf.c –come prima, ma i figli oltre che l'indice di creazione stampano anche il loro tid (Thread IDentifier).

```
osd@lica04:~/Pthread/primeProve$ dueThread-conJoinEPexitNeiThreadESelf
Sto per creare il thread 0-esimo
SONO IL MAIN e ho creato il Pthread 0-esimo con id=140592729396992
Sto per creare il thread 1-esimo
SONO IL MAIN e ho creato il Pthread 1-esimo con id=140592721004288
Thread1 partito: Hello World! Ho come identificatore 140592721004288
Thread0 partito: Hello World! Ho come identificatore 140592729396992
Pthread 0-esimo restituisce 1000
Pthread 1-esimo restituisce 1001
```