Makefile

Crea un arxiu *Makefile* per compilar tots els programes de l'examen, usant regles independents per cadascun. Inclou una regla *clean* per eliminar tots els binaris, arxius objecte i/o fitxers temporals. Els programes han de compilar-se si i només si s'han actualitzat els arxius de codi font que els composen.

Control de errors i funció Usage

Tots els programes han d'incloure un control adequat dels arguments usant la funció `Usage()` i han de controlar els errors en TOTES les crides al sistema (a excepció del write per pantalla).

1: Gestió d'arguments

Escriu un programa en C anomenat **distance.c** per calcular la *distància* entre dos nodes d'un sistema P2P. El programa rebrà com arguments dos números, *n1* i *n2*, i comprovarà que cadascun d'ells sigui més gran o igual que 1 i més petit o igual que 127. En cas que el nombre d'arguments no sigui correcte, o que algun dels números no compleixi les restriccions descrites, el programa escriurà un missatge d'error i acabarà amb *exitcode* = 255. En cas que els arguments siguin correctes, el programa escriurà per pantalla el seu PID i el valor de la *distància* entre *n1* i *n2,* i acabarà amb *exitcode* igual al citat valor. La *distància* entre dos números enters es calcula mitjançant una XOR bit a bit (*n1 ^ n2*).

Exemples d'ús:

```
$ ./distance 5 5
[PID] La distància entre 5 i 5 és 0

$ ./distance 32 34
[PID] La distància entre 32 i 34 és 2

$ ./distance 12 307
Usage: distance n1 n2, 1 =< ni =< 127

$ ./distance 23
Usage: distance n1 n2, 1 =< ni =< 127
```

2: Creació i mutació de processos

Escriu un programa en C anomenat **Ndistances.c** que calculi les *distancies* entre un node origen i una llista de possibles nodes destí. El primer argument del programa correspondrà al número del node origen i a continuació vindrà una llista de números d'almenys un element corresponent als nodes destí. Per cada número de node destí, el programa haurà de crear un nou procés i mutar-lo a **distance**, passant-li el número de node origen i el corresponent número de node destí com arguments. El programa ha d'esperar als processos fill (però sense recollir el seu codi de finalització) i acabar. Els processos fill han d'executar-se seqüencialment.

NOTA: Tal com especifica l'enunciat, el programa **Ndistances** ha de comprovar que el número d'arguments rebut sigui correcte, però no comprovarà si els arguments rebuts corresponen a un número vàlid ni si està en el rang descrit. Això ho comprova el programa **distance**.

Exemples d'ús:

```
$ ./Ndistances 1 1 2 3 4 5
[PID1] La distància entre 1 i 1 és 0
[PID2] La distància entre 1 i 2 és 3
[PID3] La distància entre 1 i 3 és 2
[PID4] La distància entre 1 i 4 és 5
[PID5] La distància entre 1 i 5 és 4

$ ./Ndistances 12
Usage: Ndistances n1 n2 […]
```

3: Gestió del codi de finalització

Copia el programa **Ndistances.c** i anomena'l **Mindistance.c**. Modifica aquest programa perquè el procés pare retorni la *distància* mínima entre el node origen i tots els nodes destí. Per això, el procés pare ha d'esperar a que acabin els processos fill i recollir la seva *distància* amb el node origen (que es retorna en el *exitcode*). Un cop obtingudes totes les *distancies*, el procés pare escriurà per pantalla la *distància* mínima, tal com es pot veure a l'exemple. Si algun dels processos fill retorna un error (255) com *exitcode*, aquest procés no es considerarà per calcular el mínim.

Exemple d'ús:

```
$ ./Mindistance 1 1 2 3 4 5
[PID1] La distància entre 1 i 1 és 0
[PID2] La distància entre 1 i 2 és 3
[PID3] La distància entre 1 i 3 és 2
[PID4] La distància entre 1 i 4 és 5
[PID5] La distància entre 1 i 5 és 4
[PID0] El procés PID1 ha retornat la distància mínima = 0
```

Qué s'ha de lliurar?

Un únic fitxer *tar.gz* amb el codi font i el Makefile. Per generar-lo podeu usar la comanda següent:

```
tar zcvf simlab1.tar.gz distance.c Ndistances.c Mindistance.c Makefile
```