Qüestionari sobre threads i concurrència

1. Explica quins avantatges i desavantatges té el paradigma de programació paral·lela (shared memory) en front al paradigma de programació distribuïda (shared nothing).

Com d'avantatges principals de la memòria compartida, ens trobem que qualsevol nombre de processos es poden comunicar alhora, ja que fan ús de la mateixa regió de memòria. A conseqüència, la comunicació és molt més ràpida que un pas de missatges, ja que un cop un dels processos ha modificat la variable, qualsevol altre procés que accedeixi a aquesta ja tindrà el valor actualitzat. Aquest avantatge també es pot convertir en desavantatge, ja que poden haver problemes de protecció de memòria.
Per altra banda, en shared nothing, cada procés es completament independent dels altres. Al no compartir recursos, elimina problema del single point of failure, per tant en cas de que un node/procés caigués, la resta del sistema continuaria funcionant.

2. Descriu tres tipus d'aplicacions reals de programació paral·lela i tres tipus d'aplicacions reals de programació distribuïda.

En el sector de la ciència, per exemple en la previsió del temps, utilitza diferents processos paral·lels per poder entendre i mapejar els patrons de temperatura, humitat, núvols… per d'aquesta forma poder obtenir un resultat conjunt.
En el sector de la banca fan molt ús, ja que tenen grans quantitats d'informació a processar, com la puntuació del crèdit de un usuari, detecció de frau… Tots aquests processos, son accelerats per GPUs, ja que aquestes poden realitzar una gran quantitat de càlculs en paral·lel.
El gaming també en fa ús, alhora de calcular diferents físiques com les col·lisions, els Fluids, reflexos… El renderitzat es una tasca paral·lela.

Per altra banda, en el cas de la programació distribuïda, ens trobem les xarxes de telecomunicacions, on diversos sistemes estan connectats i la informació d'un, esta replicada las altres. YouTube es un exemple.
P2P, el torrenting es també un cas on podem veure de forma clara, com hi ha una interconnexió de diferents sistemes sense requerir de un màster.
Cloud computing, com Kubernetes, pot haver-hi una replicació de un servei, i tots aquests compartir la informació.

5. Resultats: 100k array

	EX4 (threads)	EX5 (mem compartida)
THREADS 1	975000	1250100
THREADS 2	1434100	2718000
THREADS 8	1592300	1871300
THREADS 16	2055800	2779200
THREADS 100	10285500	9811000
THREADS 400	28609300	28563600

Com més baix el nombre de threads, la diferencia que veiem de temps és més alta. Podem veure com fins els 16 threads, la memòria compartida aconsegueix ser aproximadament el doble de ràpida que els threads.
Un cop el nombre de threads ja supera els 16, la diferencia es va reduint fins arribar el punt com als 100 i 400 threads on la diferencia es nul·la, i els resultats son “exactament” iguals.

En un principi, el procés de memòria compartida és més lent degut a que els diferents threads intenten accedir a la mateixa regió de memòria al mateix temps, i això provoca que es relentitzi en comparació amb el cas dels threads, on cada un d’ells te el seu propi array. Arriba un punt però, que el tenir un nombre tant gran de threads provoca que la diferencia sigui nul·la, ja que la creació de tants arrays “retallats” suma tant temps com el fet de fer un accés a memòria compartida.

6. Justifica quina diferència hi ha entre emprar el mètode run i el mètode start de la classe Thread.

La diferencia, es que en cas de cridar el start, es crearà un nou thread i un cop creat, la funció run, de dins del thread serà cridada.
En cas de cridar directament el run del thread, el que passarà és que ens estarem saltant la creació del thread, per tant estarem corrent el codi que hi ha dins de run en el procés principal, com si de una funció normal es tractés. És per aquest motiu, que el mètode start només pot ser cridat un cop per classe de thread que tinguem, mentre que el run el podem cridar tants cops com vulguem.

8. Compara el temps d'execució de l’exercici anterior amb una ordenació seqüencial. Justifica els resultats.

10000 elements:

Temps Threads: 165213500
Temps Seqüencial: 481200

Podem veure que al fer el merge sort amb threads és molt més lent que al fer-ho seqüencialment. Això es degut a que el crear un thread es una extra tasca que estem fent, que consumeix un extra de temps. El fer un thread, en alguns casos pot ajudar-nos a realitzar una tasca de forma més ràpidament, ja que aprofitem el paral·lelisme, però en cas de una tasca tant senzilla com un merge sort, al final només perdem temps.
Per tant, el problema en aquest cas es la creació de masses threads.