Jaime Martínez Sánchez

Práctica 3

1.- Escribir el archivo de descripción de trabajos necesario para simular un programa de ordenación de un vector, con diversos parámetros de configuración:

Para ello, dentro de la carpeta *dlx*, se ha creado un fichero *dlxArguments.cmd,* donde estarán los trabajos. Se ha utilizado $(Process) para mostrar las distintas salidas o los distintos ficheros de error. Por último, en el parámetro *arguments* escribimos cada una de las instrucciones, seguidas de la instrucción *queue.* Por último, cambiamos el *input* a *ordena3.s,* y ejecutamos las dos últimas instrucciones con este fichero.

Para comprobar que todo ha funcionado correctamente, se consultan los distintos ficheros de salida generados. Abriendo uno de ellos, se encuentra el siguiente resultado:

```
Mem[   0]:    0 Mem[   8]:    0 Mem[  16]:    0 Mem[  24]:    0
Mem[  32]:    0 Mem[  40]:    0 Mem[  48]:    0 Mem[  56]:    0
Mem[  64]:    0 Mem[  72]:    0 Mem[  80]:    0 Mem[  88]:    0
Mem[  96]:    0 Mem[ 104]:    0 Mem[ 112]:    0 Mem[ 120]:    0
Mem[ 128]:    0 Mem[ 136]:    1 Mem[ 144]:    1 Mem[ 152]:    1
Mem[ 160]:    1 Mem[ 168]:    1 Mem[ 176]:    1 Mem[ 184]:    1
Mem[ 192]:    1 Mem[ 200]:    1 Mem[ 208]:    1 Mem[ 216]:    1
Mem[ 224]:    1 Mem[ 232]:    1 Mem[ 240]:    1 Mem[ 248]:    1
Mem[ 256]:    1 Mem[ 264]:    1 Mem[ 272]:    1 Mem[ 280]:    1
Mem[ 288]:    1 Mem[ 296]:    1 Mem[ 304]:    1 Mem[ 312]:    1
Mem[ 320]:    1 Mem[ 328]:    1 Mem[ 336]:    1 Mem[ 344]:    1
Mem[ 352]:    1 Mem[ 360]:    1 Mem[ 368]:    1 Mem[ 376]:    1
Mem[ 384]:    2 Mem[ 392]:    2 Mem[ 400]:    2 Mem[ 408]:    2
Mem[ 416]:    2 Mem[ 424]:    2 Mem[ 432]:    2 Mem[ 440]:    2
Mem[ 448]:    2 Mem[ 456]:    2 Mem[ 464]:    2 Mem[ 472]:    2
Mem[ 480]:    2 Mem[ 488]:    2 Mem[ 496]:    2 Mem[ 504]:    2
Mem[ 512]:    2 Mem[ 520]:    2 Mem[ 528]:    2 Mem[ 536]:    2
Mem[ 544]:    2 Mem[ 552]:    2 Mem[ 560]:    2 Mem[ 568]:    2
Mem[ 576]:    2 Mem[ 584]:    2 Mem[ 592]:    2 Mem[ 600]:    2
Mem[ 608]:    2 Mem[ 616]:    2 Mem[ 624]:    2 Mem[ 632]:    2
Mem[ 640]:    2 Mem[ 648]:    2 Mem[ 656]:    2 Mem[ 664]:    2
Mem[ 672]:    3 Mem[ 680]:    3 Mem[ 688]:    3 Mem[ 696]:    3
Mem[ 704]:    3 Mem[ 712]:    3 Mem[ 720]:    3 Mem[ 728]:    3
Mem[ 736]:    3 Mem[ 744]:    3 Mem[ 752]:    3 Mem[ 760]:    3
Mem[ 768]:    3 Mem[ 776]:    3 Mem[ 784]:    3 Mem[ 792]:    3
Mem[ 800]:    3 Mem[ 808]:    3 Mem[ 816]:    3 Mem[ 824]:    3
Mem[ 832]:    3 Mem[ 840]:    3 Mem[ 848]:    3 Mem[ 856]:    3
Mem[ 864]:    3 Mem[ 872]:    3 Mem[ 880]:    3 Mem[ 888]:    3
Mem[ 896]:    3 Mem[ 904]:    3 Mem[ 912]:    3 Mem[ 920]:    3
```

Como se puede ver, el vector se encuentra ordenado en memoria.

2.- Repartir el cálculo de integrales definidas entre todos los nodos de Condor. Para ello, se usan 50 subintervalos, y se suman los resultados obtenidos.

Utilizando el fichero"prep.c" proporcionado, se generan los 50 ficheros que se deben de sumar para obtener el resultado final:

```
cos16@cac1:~/prac3-condor/dagman> ls
in.0   in.12  in.16  in.2   in.23  in.27  in.30  in.34  in.38  in.41  in.45  in.49  in.8
   sum.c
in.1   in.13  in.17  in.20  in.24  in.28  in.31  in.35  in.39  in.42  in.46  in.5   in.9
   sum.condor
in.10  in.14  in.18  in.21  in.25  in.29  in.32  in.36  in.4   in.43  in.47  in.6   intsin
r
in.11  in.15  in.19  in.22  in.26  in.3   in.33  in.37  in.40  in.44  in.48  in.7   intsin.c
```

Para obtener los valores de salida, se va a crear el siguiente fichero "intsin.condor" que permita lanzar un trabajo por cada fichero de entrada:

```
cos16@cac1:~/prac3-condor/dagman> cat intsin.condor
universe = vanilla
executable = intsin

log=intsin.log
error=intsin.error
input=in.$(Process)
output=out.$(Process)
queue 50
```

A continuación, utilizando el fichero "sum.c", se va a sumar los archivos que comienzan por "out", obteniéndose lo siguiente en la salida:

```
cos16@cac1:~/prac3-condor/dagman> ./sum 50 out
0.000016
```

El resultado es lógico, por lo que se concluye que el programa ha funcionado correctamente.