# PRÁCTICA 1: MPI

En los sistema UNIX se dispone de un función `crypt` que permite realizar un HASH de una cadena de hasta 8 caracteres empleando el algoritmo DES.

La función `crypt` necesita dos parámetros:

- El primero es la cadena que queremos encriptar.
- El segundo es una cadena (`salt`) de dos caracteres (del conjunto a-zA-Z0-9./) y se utiliza para que una misma cadena a encriptar, tenga 4096 diferentes resultados.

El resultado retornado, es una cadena de 13 caracteres, siendo los dos primeros el `salt` pasado como parámetro.

```
CRYPT(3)                                        Linux Programmer's Manual                                        CRYPT(3)

NAME
       crypt, crypt_r - password and data encryption
SYNOPSIS
       #define _XOPEN_SOURCE       /* See feature_test_macros(7) */
       #include <unistd.h>

       char *crypt(const char *key, const char *salt);

       #define _GNU_SOURCE         /* See feature_test_macros(7) */
       #include <crypt.h>

       char *crypt_r(const char *key, const char *salt,
                     struct crypt_data *data);

       Link with -lcrypt.
DESCRIPTION
       crypt()  is  the  password  encryption  function.   It  is  based on the Data Encryption Standard algorithm with variations
       intended (among other things) to discourage use of hardware implementations of a key search.

       key is a user's typed password.

       salt is a two-character string chosen from the set [a–zA–Z0–9./].  This string is used to perturb the algorithm in  one  of
       4096 different ways.
```

El siguiente programa muestra el uso de la función `crypt` para generar la clave encriptada correspondiente al parámetro dado como entrada.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define _XOPEN_SOURCE

#include <unistd.h>
#include <crypt.h>
void main(int argc,char **argv)
{
        printf("%s\n",crypt(argv[1],"aa"));
}
```

Un ejemplo de ejecución de este programa es:

```
./crypt "00125414"
aaTrgM6tVLhas
```

Supongamos que disponemos de la clave encriptada "`aaTrgM6tVLhas`" y que sabemos que corresponde a una clave de 8 caracteres y que solamente contiene números. Si se desease descubrir a que valor corresponde se podrían generar números aleatorios de ("00000000" a "99999999").

```
void main(int argc,char **argv)
{
        char con[9];
        unsigned long long intentos;

        srand(1);
        do
        {
                sprintf(con,"%08d",rand()%100000000);
                intentos++;
                if (strcmp(crypt(con,"aa"),argv[1])==0)
                {
                        printf("Encontrada: %s->%s (%lld)\n",argv[1],con, intentos);
                        break;
                }
        } while (1);
}
```

Un ejemplo de ejecución de este programa es:

```
./decrypt "aaTrgM6tVLhas"
Encontrada: aaTrgM6tVLhas->00125414 (3183022619)
```

## SE PIDE:

- **Realizar un programa en MPI que permita distribuir lo visto anteriormente, para localizar una lista de claves, cuyas cadenas encriptadas creará y solamente las conoce el proceso 0. Para ello:**
  - Se tendrá un proceso encargado de la E/S, que podrá ser o no calculador, habrá que elegir la mejor opción, que deberá:
    - Generar la lista de claves a repartir (dependiendo de la longitud de la clave tardará más o menos en encontrarlas)
    - Comunicar al resto de procesos de una en una las claves a descubrir.
    - En el caso de no haber más claves aplicar la política de ayudar al compañero en vez de quedarse sin hacer nada, para ello se le asignará una clave, aunque ya esté siendo buscada por otro proceso, de manera que puedan haber varios procesos buscando la misma clave, pero nunca, un proceso buscando una clave que ya ha sido encontrada. Esto implica que debe haber un mecanismo para detener una búsqueda de una clave que ha sido encontrada por otro proceso.
    - Recolectar los datos y mostrar los resultados: clave encontrada, quién la ha encontrado, cuantos intentos se han necesitado, tiempo empleado, etc.. y a partir de ahí calcular el número de claves comprobadas por segundo.
    - Mostrar las estadísticas finales.
  - N procesos calculadores que:
    - Esperarán a que el proceso de E/S les indique la clave a descubrir
    - A través de la **generación de números aleatorios** irá probando claves, dentro del rango válido (por ejemplo: 7 dígitos, 0000000-9999999).
    - Una vez encontrada una clave por un proceso lo notificará al proceso 0, que se encargará de mostrarlo por pantalla y de asignarle un nuevo trabajo.
    - Los procesos podrán devolver los resultados como mejor se considere.
    - De alguna manera deberán mandar al proceso 0 el número de claves que han intentado.
  - El reparto de tareas se deberá formular para que sea lo más equitativo posible: que unos procesos no retrasen a otros, que no hayan procesos en espera, etc.
  - Todos los procesos deben terminar, no se pueden quedar procesos colgados ni comunicaciones incompletas.

- **Una vez realizado el programa, realizar un estudio de rendimiento:**
  - Tomar datos de rendimiento en 1,2 y 4 máquinas, con el siguiente número de procesos: 1(secuencial),2,3,4,5,6,7,8,16,32,64. Se valorará, el estudio en más máquinas y con más procesos.
  - Realizar un estudio sobre el resultado de aumentar el número de procesos y máquinas: como se reduce el tiempo o no, hasta un cierto momento, de qué manera aumenta el rendimiento, comparar con el rendimiento ideal.
  - Es recomendable que el tiempo de ejecución en 1 proceso sea alto para que la mejora sea palpable.

## SE DEBERÁ ENTREGAR (EN LAS FECHAS INDICADAS EN STUDIUM):

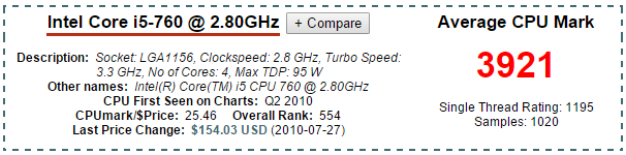
- Código fuente del programa realizado en un único archivo.
- Presentación empleada en el Seminario: Aspectos relevantes, Comunicaciones empleadas, Pruebas realizadas con análisis de resultados, con el mejor rendimiento cuanto se tardaría en encontrar una clave que contenga todos los caracteres posibles, conclusiones…

## NOTA:

- Se deberán realizar pruebas en las que el resultado de la distribución sea palpable, es decir, que en secuencial los tiempos de ejecución sean grandes, y se observe la mejora de rendimiento
- El tiempo que se tarda en descubrir una clave depende de la clave en sí y del generador de números aleatorios.
- Realizar pruebas tanto en una máquina como en varias.
- Hay que tener cuidado con los tipos de datos empleados y que se transmiten
- La práctica no trata de optimizar algoritmos ni de emplear las opciones de optimización del compilador, trata de emplear fuerza bruta.
- Si se usan números aleatorios, los cálculos dependen de la semilla, en distintos proceso habrá que iniciar con distinta semilla.
- Se deberá subir al Studium en un único archivo, un único componente del grupo.
- La presentación de la práctica se realizará en la sesión marcada como Seminario, teniendo que subir la presentación en las fechas marcadas en Studium.
- Cualquier modificación del enunciado de publicará en Studium.
- La detección de copia parcial o total de la práctica conllevará la suspensión de las prácticas, y por tanto de la asignatura.

**Ejemplo de resultados obtenidos (ES SOLAMENTE UN EJEMPLO: Dependiendo de la manera de cómo se implemente los resultados pueden variar, pero las tendencias en ganancia de rendimiento siempre irán conforme al hardware empleado. Las gráficas mostradas son solamente orientativas).**

*Máquina 1 empleada:*




- 4GB de memoria
- Ubuntu 16.04 (64 bits)
- OpenMPI 1.4.3

La salida por pantalla corresponde a la generación de 20 claves de 7 cifras numérica y el uso de 4 procesos para descubrirlas, en la que se puede ver la lista de claves encriptadas y quién la ha encontrado, así como la clave asociada a cada proceso, cada vez que se encuentra una clave, y por último las estadísticas finales

```
Proceso 0 en desktopUbuntu con semilla 100
Proceso 1 en desktopUbuntu con semilla 200
Proceso 2 en desktopUbuntu con semilla 300
Proceso 3 en desktopUbuntu con semilla 400
>--CLAVES GENERADAS CON SEMILLA 9999--<
RAND_MAX: 2147483647
00) 1791766 aaYEwvI9dPiGQ
01) 9106993 aaMBf0eVD71Uw
02) 3091667 aabGfehjc8Eyw
03) 2998163 aalmzjcSBw.AA
04) 8737059 aaHkZjL3q1BJ6
05) 3911251 aaD1XQa53JNC2
06) 9869014 aawqqxafwE7V2
07) 2348195 aaDBiTIjfHFHc
08) 3210938 aapRU2hi4BREU
09) 7320440 aazbZf3wvVmlg
10) 9733890 aaMdZI43ND5Lg
11) 8487791 aaeNwMptpxNdk
12) 5897544 aazCna2qaWqFc
13) 0581076 aauWE4vm5lKJ6
14) 2370933 aaCFHmQxIVcPQ
15) 2618859 aa4uOVhCxsgaA
16) 7533745 aaHj6PdgGdjTM
17) 6446081 aao8EaQ5W.ZX.
18) 9036669 aaC/sTptMJlzo
19) 3786927 aaN2G8qrDoTJ2

++++++++++++++++++++++++++
COMIENZA LA BUSQUEDA
PROC 00: aaYEwvI9dPiGQ (0)
PROC 01: aaMBf0eVD71Uw (1)
PROC 02: aabGfehjc8Eyw (2)
PROC 03: aalmzjcSBw.AA (3)
```

```
>----------------------<
000) aaYEwvI9dPiGQ-> (-1)
001) aaMBf0eVD71Uw-> (-1)
002) aabGfehjc8Eyw-> (-1)
003) aalmzjcSBw.AA->2998163 (3)
004) aaHkZjL3q1BJ6-> (-1)
005) aaD1XQa53JNC2-> (-1)
006) aawqqxafwE7V2-> (-1)
007) aaDBiTIjfHFHc-> (-1)
008) aapRU2hi4BREU-> (-1)
009) aazbZf3wvVmlg-> (-1)
010) aaMdZI43ND5Lg-> (-1)
011) aaeNwMptpxNdk-> (-1)
012) aazCna2qaWqFc-> (-1)
013) aauWE4vm5lKJ6-> (-1)
014) aaCFHmQxIVcPQ-> (-1)
015) aa4uOVhCxsgaA-> (-1)
016) aaHj6PdgGdjTM-> (-1)
017) aao8EaQ5W.ZX.-> (-1)
018) aaC/sTptMJlzo-> (-1)
019) aaN2G8qrDoTJ2-> (-1)
PROC 00: aaYEwvI9dPiGQ (0)
PROC 01: aaMBf0eVD71Uw (1)
PROC 02: aabGfehjc8Eyw (2)
PROC 03: aaHkZjL3q1BJ6 (4)
```

```
>-----------------<
000) aaYEwvI9dPiGQ-> (-1)
001) aaMBf0eVD71Uw-> (-1)
002) aabGfehjc8Eyw->3091667 (2)
003) aalmzjcSBw.AA->2998163 (3)
004) aaHkZjL3q1BJ6-> (-1)
005) aaD1XQa53JNC2-> (-1)
006) aawqqxafwE7V2-> (-1)
007) aaDBiTIjfHFHc-> (-1)
008) aapRU2hi4BREU-> (-1)
009) aazbZf3wvVmlg-> (-1)
010) aaMdZI43ND5Lg-> (-1)
011) aaeNwMptpxNdk-> (-1)
012) aazCna2qaWqFc-> (-1)
013) aauWE4vm5lKJ6-> (-1)
014) aaCFHmQxIVcPQ-> (-1)
015) aa4uOVhCxsgaA-> (-1)
016) aaHj6PdgGdjTM-> (-1)
017) aao8EaQ5W.ZX.-> (-1)
018) aaC/sTptMJlzo-> (-1)
019) aaN2G8qrDoTJ2-> (-1)
PROC 00: aaYEwvI9dPiGQ (0)
PROC 01: aaMBf0eVD71Uw (1)
PROC 02: aaD1XQa53JNC2 (5)
PROC 03: aaHkZjL3q1BJ6 (4)
```

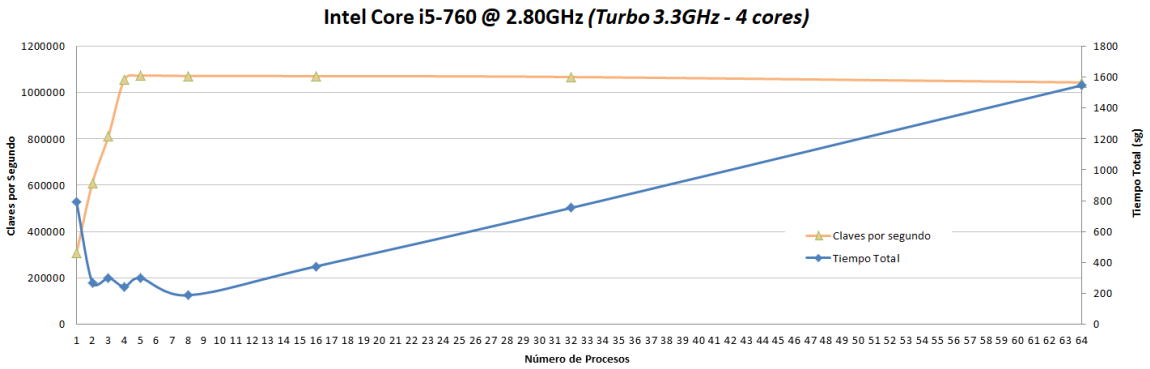
```
>--------------------------------<
000) aaYEwvI9dPiGQ->1791766 (0)
001) aaMBf0eVD71Uw->9106993 (1)
002) aabGfehjc8Eyw->3091667 (2)
003) aalmzjcSBw.AA->2998163 (3)
004) aaHkZjL3q1BJ6->8737059 (3)
005) aaD1XQa53JNC2->3911251 (2)
006) aawqqxafwE7V2->9869014 (2)
007) aaDBiTIjfHFHc->2348195 (3)
008) aapRU2hi4BREU->3210938 (3)
009) aazbZf3wvVmlg->7320440 (2)
010) aaMdZI43ND5Lg->9733890 (1)
011) aaeNwMptpxNdk->8487791 (3)
012) aazCna2qaWqFc->5897544 (3)
013) aauWE4vm5lKJ6->0581076 (3)
014) aaCFHmQxIVcPQ->2370933 (0)
015) aa4uOVhCxsgaA->2618859 (1)
016) aaHj6PdgGdjTM->7533745 (3)
017) aao8EaQ5W.ZX.-> (-1)
018) aaC/sTptMJlzo-> (-1)
019) aaN2G8qrDoTJ2->3786927 (3)
PROC 00: aao8EaQ5W.ZX. (17)
PROC 01: aaC/sTptMJlzo (18)
PROC 02: aao8EaQ5W.ZX. (17)
PROC 03: aao8EaQ5W.ZX. (17)
```

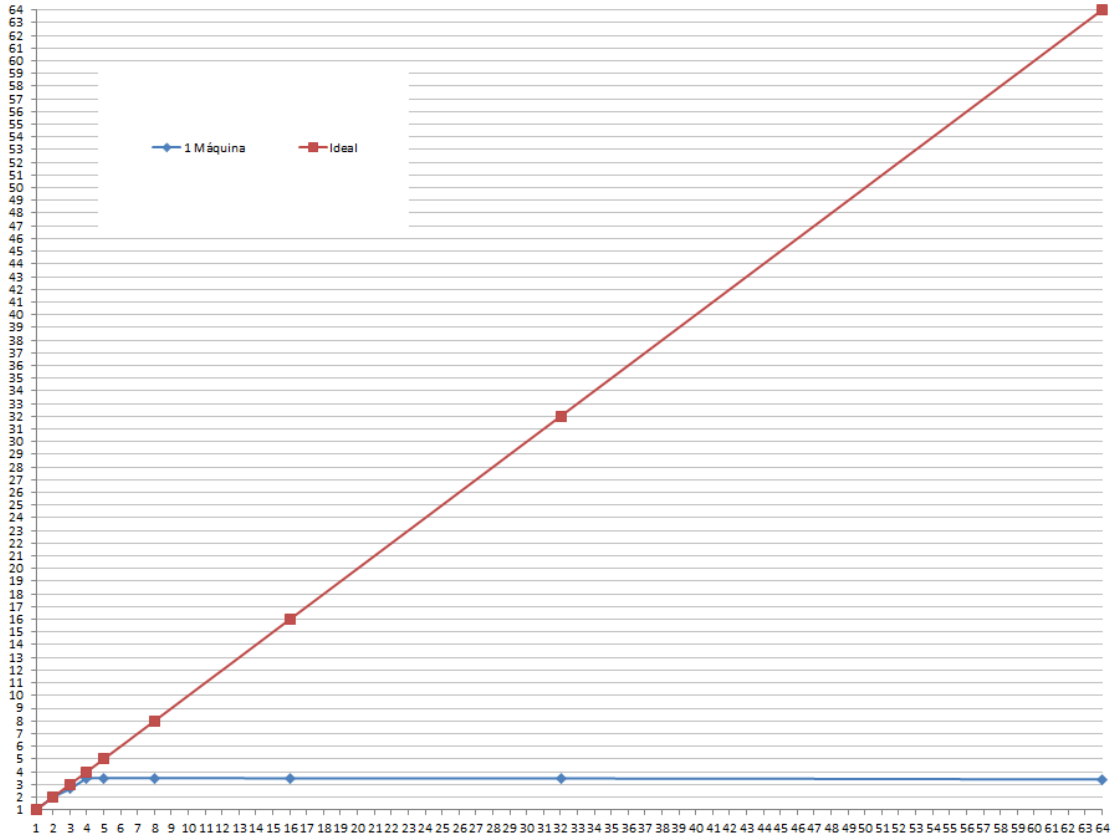
```
>--------------------------------<
000) aaYEwvI9dPiGQ->1791766 (0)
001) aaMBf0eVD71Uw->9106993 (1)
002) aabGfehjc8Eyw->3091667 (2)
003) aalmzjcSBw.AA->2998163 (3)
004) aaHkZjL3q1BJ6->8737059 (3)
005) aaD1XQa53JNC2->3911251 (2)
006) aawqqxafwE7V2->9869014 (2)
007) aaDBiTIjfHFHc->2348195 (3)
008) aapRU2hi4BREU->3210938 (3)
009) aazbZf3wvVmlg->7320440 (2)
010) aaMdZI43ND5Lg->9733890 (1)
011) aaeNwMptpxNdk->8487791 (3)
012) aazCna2qaWqFc->5897544 (3)
013) aauWE4vm5lKJ6->0581076 (3)
014) aaCFHmQxIVcPQ->2370933 (0)
015) aa4uOVhCxsgaA->2618859 (1)
016) aaHj6PdgGdjTM->7533745 (3)
017) aao8EaQ5W.ZX.-> (-1)
018) aaC/sTptMJlzo->9036669 (1)
019) aaN2G8qrDoTJ2->3786927 (3)
PROC 00: aao8EaQ5W.ZX. (17)
PROC 01: aao8EaQ5W.ZX. (17)
PROC 02: aao8EaQ5W.ZX. (17)
PROC 03: aao8EaQ5W.ZX. (17)
```

```
>---CLAVES DESENCRIPTADAS----------<
00) aaYEwvI9dPiGQ->1791766 (0)
01) aaMBf0eVD71Uw->9106993 (1)
02) aabGfehjc8Eyw->3091667 (2)
03) aalmzjcSBw.AA->2998163 (3)
04) aaHkZjL3q1BJ6->8737059 (3)
05) aaD1XQa53JNC2->3911251 (2)
06) aawqqxafwE7V2->9869014 (2)
07) aaDBiTIjfHFHc->2348195 (3)
08) aapRU2hi4BREU->3210938 (3)
09) aazbZf3wvVmlg->7320440 (2)
10) aaMdZI43ND5Lg->9733890 (1)
11) aaeNwMptpxNdk->8487791 (3)
12) aazCna2qaWqFc->5897544 (3)
13) aauWE4vm5lKJ6->0581076 (3)
14) aaCFHmQxIVcPQ->2370933 (0)
15) aa4uOVhCxsgaA->2618859 (1)
16) aaHj6PdgGdjTM->7533745 (3)
17) aao8EaQ5W.ZX.->6446081 (1)
18) aaC/sTptMJlzo->9036669 (1)
19) aaN2G8qrDoTJ2->3786927 (3)
>----CLAVES ENCONTRADAS POR PROCESO----<
00) 2
01) 5
02) 4
03) 9
>----LLAMADAS A RAND Y CRYPT------<
00) 65436123
01) 65370290
02) 60462042
03) 65023327
>---------<
>----ESTADISTICAS TOTALES-------<
Intentos acumulados: 256291782
Segundos por Clave : 0.000000947824437
Numero Procesos: 4
Tiempo Procesamiento: 242.919614

>----------------FIN----------------<
```

Prueba realizada para 20 claves de 6 cifras numéricas en una y dos máquinas con varios procesadores




*Máquina: 2 empleadas (AULA SUN en una y dos máquinas con balanceo automático):*

**Intel Core i7-4770 @ 3.40GHz** + Compare

**Description:** *Socket: LGA1150, Clockspeed: 3.4 GHz, Turbo Speed: 3.9 GHz, No of Cores: 4 (2 logical cores per physical), Max TDP: 84 W*
**Other names:** *Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz*
**CPU First Seen on Charts:** Q2 2013
**CPUmark/$Price:** 32.03 **Overall Rank:** 87
**Last Price Change:** $306.99 USD (2013-06-04)

**Average CPU Mark**

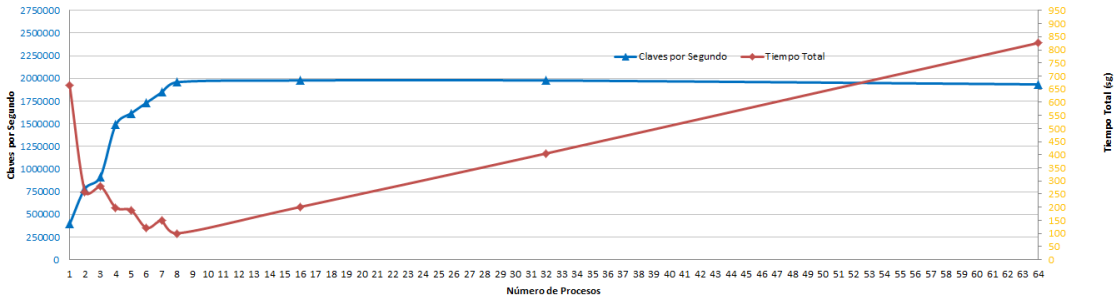
**9832**

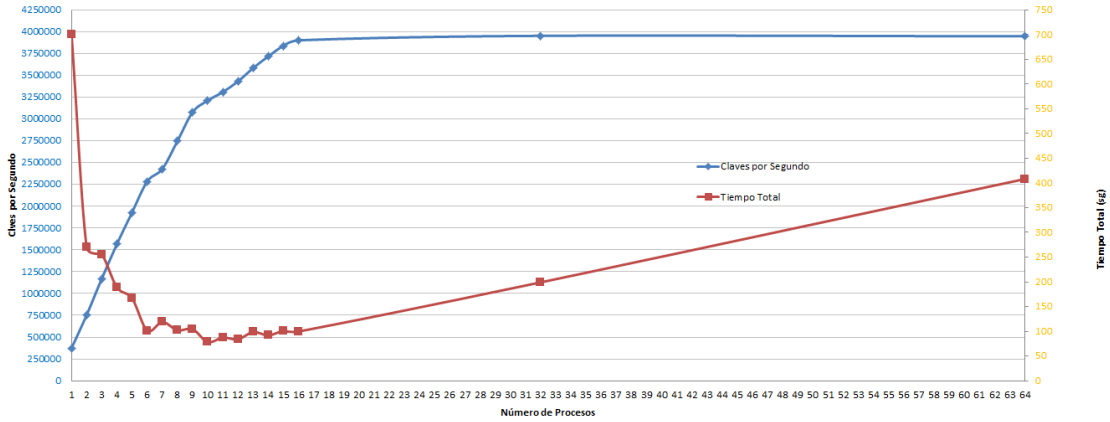
Single Thread Rating: 2232
Samples: 4034

- 8GB de memoria
- Debian 8 (64 bits)
- OpenMPI 1.4.5

Prueba realizada para 20 claves de 7 cifras numéricas en una y dos máquinas con varios procesadores

**Intel Core i7-4770 @ 3.40GHz** (*Cores: 4 (2 logical cores per physical)*)




**2 x Intel Core i7-4770 @ 3.40GHz** (*Cores: 4 (2 logical cores per physical)*)