Luis Guillermo Rivera
Carlo Alessandro Santoro

Actividad 6

```
/*
 * Para compilar hay que agregar la librería matemática
 *         gcc -o matprimos matprimos.c -lm
 *                 gcc -o matprimos matprimos.c -lm -lpthread
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <sys/time.h>
#include <pthread.h>


#define SIZE 4000
#define NUM_THREADS 4

#define INICIAL 900000000
#define FINAL 1000000000

int mat[SIZE][SIZE];

typedef struct {
   int start;
   int end;
} threadData;

void initmat(int mat[][SIZE]);
void printnonzeroes(int mat[SIZE][SIZE]);
int isprime(int n);
void* findNonPrimes(void* x);

int main()
{
        long long start_ts;
        long long stop_ts;
        long long elapsed_time;
        long lElapsedTime;
        struct timeval ts;

        // Inicializa la matriz con números al azar
        initmat(mat);
```

```
        gettimeofday(&ts, NULL);
        start_ts = ts.tv_sec; // Tiempo inicial

        // Eliminar de la matriz todos los números que no son primos
        // Esta es la parte que hay que paralelizar
        pthread_t arrThreads[NUM_THREADS];
    threadData arrThrData[NUM_THREADS];
    int rowsPorThr = SIZE/NUM_THREADS;

    for (int i = 0; i < NUM_THREADS; i++) {
        arrThrData[i].start = i * rowsPorThr;
        arrThrData[i].end = (i+1) * rowsPorThr;
        pthread_create(&arrThreads[i], NULL, findNonPrimes, (void*)&arrThrData[i]);
    }

    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(arrThreads[i], NULL);
    }

        // Hasta aquí termina lo que se tiene que hacer en paralelo
        gettimeofday(&ts, NULL);
        stop_ts = ts.tv_sec; // Tiempo final
        elapsed_time = stop_ts - start_ts;


        printnonzeroes(mat);
        printf("Tiempo total, %lld segundos\n",elapsed_time);
}

void* findNonPrimes(void* x){
    threadData* thr = (threadData*)x;
    int start = thr->start;
    int end = thr->end;

    for(int i=start;i<end;i++)
                for(int j=0;j<SIZE;j++)
                        if(!isprime(mat[i][j]))
                            mat[i][j]=0;

    pthread_exit(NULL);
}

void initmat(int mat[][SIZE])
{
        int i,j;

        srand(getpid());
```

```
	for(i=0;i<SIZE;i++)
		for(j=0;j<SIZE;j++)
			mat[i][j]=INICIAL+rand()%(FINAL-INICIAL);
}

void printnonzeroes(int mat[SIZE][SIZE])
{
	int i,j;

	for(i=0;i<SIZE;i++)
		for(j=0;j<SIZE;j++)
			if(mat[i][j]!=0)
				printf("%d\n",mat[i][j]);
}


int isprime(int n)
{
	int d=3;
	int prime=n==2;
	int limit=sqrt(n);

	if(n>2 && n%2!=0)
	{
		while(d<=limit && n%d)
			d+=2;
		prime=d>limit;
	}
	return(prime);
}
```

¿ Que aprendiste?

Aprendimos a hacer que se ejecuten programas concurrentes con hilos para aprovechar el uso de los procesadores, principalmente el aprovechamiento de la arquitectura multicore. Hicimos dos casos, uno específico para la máquina virtual de lubuntu que tiene 4 cores y otra más escalable para arquitecturas con más cores.

Principalmente usamos la de 4 hilos puesto que crear tantos hilos era innecesario porque ya sabíamos la cantidad de cores, y aunque tardaba lo mismo el planificador a nivel kernel se lleva menos carga.