# Memoria Práctica 1. SOPER.

## Ejercicio 13.

### Apartado a)

En este apartado buscamos implementar un proceso minero que, dado un “target”, un numero de rondas y un numero de hilos encuentre un valor para una “x” tal que se cumpla f(x) = “target” utilizando el numero dado de hilos y sustituyendo el valor “target” dado por el valor resultado en cada una de las rondas. Todo ello se realizará de la siguiente manera:

Para comenzar, en la función principal  declaramos un bucle for que haga tantas iteraciones como rondas se haya indicado. Dentro de cada iteración se generará un numero dado de hilos, asignandole a cada uno una información que se guardará en una estructura de datos y se llama a la función pthread_create, a la cual se le pasa como argumento la informacion de cada hilo y la función “search_range” la cual, con la información que se guarda en la estructura limita el rango de búsqueda del hilo para hallar el valor que resuelva la ecuación.

Después se hará la llamada a la función pthread_join tantas veces como sea necesario para finalizar todos los hilos creados.

Luego se hace  una busqueda para encontrar el hilo que ha resuelto la ecuación y una vez encontrado se hacen dos cosas:

- La asociación que hace que el “target” de la siguiente ronda sea el resultado encontrado en la ronda actual
- Se le envía mediante un pipe al proceso monitor la información del hilo (aunque para comprobar la solución solo necesita los valores “target” y “result”)

Tras enviar los resultados a proceso monitor se leen los reslutados que este le devuelve al proceso minero mediante otro pipe, y en función de el resultado obtenido, el minero dirá si a solución ha sido o no invalidada

El funcionamiento de la función “search_range" mencionada enteriormente es el siguiente: primero delimita la zona sobre la que va a actuar y hace un bucle recorriendo dicha fraccion.

```
/**
 * @brief Realiza las rondas de minado con el objetivo de resolver el problema dado
 *
 * Para cada ronda del proceso, divide el espacio de búsqueda en tantos hilos
 * como se haya especificado. Cuando un hilo encuentra la solución, terminan todos
 * los hilos y se devuelve la solución al hilo principal.
 *
 * @param fd_write int
 * @param fd_read int
 * @param target long int; Objetivo del problema inicial que debe resolver.
 * @param num_rounds int; Número de rondas de minado
 * @param num_threads int; Número de hilos que debe utilizar.
 *
 * @return Returns EXIT_SUCCESS if there are not errors, EXIT_FAILURE otherwise.
 **/
int minero(int fd_write, int fd_read, long int target, int num_rounds, int num_threads)
{

    int buffer;
    int i, j, res;
```

```
    pthread_t threads[num_threads];
    ThreadData thread_data[num_threads];

    for (i = 0; i < num_rounds; i++) /* Recorremos todas las rondas */
    {
        for (int j = 0; j < num_threads; j++) /* Recorremos los hilos */
        {
            thread_data[j].thread_id = j;
            thread_data[j].num_threads = num_threads;
            thread_data[j].target = target;

            /* Creamos hilo: */
            pthread_create(&threads[j], NULL, search_range, &thread_data[j]);
        }

        for (int j = 0; j < num_threads; j++) /* Recorremos los hilos */
        {
            /* Unimos el hilo: */
            pthread_join(threads[j], NULL);
        }

        int found = 0;
        for (j = 0; j < num_threads; j++) /* Recorremos los hilos */
        {
            if (thread_data[j].result != -1) /* Esta bien */
            {
                /* Escribe la info. del hilo en el pipe: */
                res = write(fd_write, &thread_data[j], sizeof(thread_data[j]));
                if (res == -1) /* Ha ocurrido un error */
                {
                    printf("ERROR ESCRIBIENDO FD1\n");
                    perror("write");
                    exit(EXIT_FAILURE);
                }

                /* Se le asigna a target el valor del resultado encontrado para la siguiente ronda: */
                target = thread_data[j].result;
                found = 1;
                break;
            }
        }

        if (!found) /* Ha ocurrido un error */
        {
            printf("Target not found!\n");
            exit(EXIT_FAILURE);
        }
        /* Pasa la info. del pipe al buffer: */
        res = read(fd_read, &buffer, sizeof(int));

        if (res == -1) /* Ha habido un error */
        {
            printf("ERROR LEYENDO FD2\n");
            perror("read");
            exit(EXIT_FAILURE);
        }
        if (buffer == 1) /* Ha habido un error */
        {
            printf("The solution has been invalidated\n");
            exit(EXIT_FAILURE);
        }
    }
    return EXIT_SUCCESS;
}

/**
 * @brief Rángo de búsqueda
 *
```

```
 *
 * @param data void
 **/
void *search_range(void *data)
{
    ThreadData *thread_data = (ThreadData *)data;

    long int start = (long int)thread_data->thread_id * (long int)POW_LIMIT / (long int)thread_data->num_threads;
    long int end = (long int)(thread_data->thread_id + 1) * (long int)POW_LIMIT / (long int)thread_data->num_threads;

    for (long int i = start; i < end; i++) /* Se recorre el hilo en la región asignada */
    {
        if (pow_hash(i) == thread_data->target) /* Comprobamos si se cumple */
        {
            thread_data->result = i;
            return NULL;
        }
    }

    thread_data->result = -1;
    return NULL;
}
```

## Apartado b)

En este apartado tenemos como objetivo generar los procesos que van a ejecutar todo el ejercicio.

Para empezar hacemos un primer fork() el cual generará un proceso hijo que nace del padre. En el hijo he inicializado los pipes que voy a utilizar mas adelante y realizo un nuevo fork() que, nuevamente creará un proceso hijo (nieto con respecto al proceso inicial).

En el primer proceso hijo generado llamaremos una función “llama_minero” la cual llamará a la función minera del apartado a), cerrará los extremos oportunos de los pipes y espera a que termine su proceso hijo (monitor) comprobando su estado de finalizacíon.

En el segundo proceso hijo generado (nieto) se llamará a la función “llama_monitor” la cuan hace una llamada a la función monitor y nuevamente cerrará los extremos oportunos de los pipes

Finalmente en el proceso padre espera a que termine su proceso hijo (minero) y comprueba ta,bien u estado de finalización.

```
/**
 * @brief Llama a monitor
 *
 * Cierra los pipes y llama al monitor
 *
 * @param fd_write int
 * @param fd_read int
 *
 * @return Returns EXIT_SUCCESS if there are not errors, EXIT_FAILURE otherwise.
 **/
void llama_monitor(int fd_read[2], int fd_write[2])
{
    int mon;

    /* Cierra los pipes: */
    close(fd_read[1]);
    close(fd_write[0]);

    /* Llama a monitor: */
    mon = monitor(fd_write[1], fd_read[0]);
```

```
    /* Cierra los pipes: */
    close(fd_write[1]);
    close(fd_read[0]);

    exit(mon);
}

/**
 * @brief Llama a minero
 *
 * Cierra los pipes, llama al minero y comprueba el estado
 * de finalización.
 *
 * @param fd_write int
 * @param fd_read int
 * @param pid int; Id del proceso.
 * @param target long int; Objetivo del problema inicial que debe resolver.
 * @param num_rounds int; Número de rondas de minado que debe realizar.
 * @param num_threads int; Número de hijos que debe utilizar.
 *
 * @return Returns EXIT_SUCCESS if there are not errors, EXIT_FAILURE otherwise.
 **/
void llama_minero(int fd_read[2], int fd_write[2], int pid, int target, int num_rounds, int num_threads)
{

    int min, status;

    /* Cierra los pipes: */
    close(fd_write[0]);
    close(fd_read[1]);

    /* Llama a minero: */
    min = minero(fd_write[1], fd_read[0], target, num_rounds, num_threads);

    /* Cierra los pipes: */
    close(fd_write[1]);
    close(fd_read[0]);

    /* Wait del pid: */
    waitpid(pid, &status, 0);

    /* Comprobamos el valor de status: */
    if (WIFEXITED(status))
    {
        printf("Monitor exited with status %d\n", WEXITSTATUS(status));
    }
    else
    {
        printf("Monitor exited unexpectedly\n");
    }

    exit(min);
}

/**
 * @brief Comprueba el estado de finalización del minero
 *
 * @param pid int; Id del proceso
 *
 * @return Returns EXIT_SUCCESS if there are not errors, EXIT_FAILURE otherwise.
 **/
void llamada_padre(int pid)
{
    int status;

    /* Wait del pid: */
    waitpid(pid, &status, 0);
```

```
    /* Comprobamos el valor del status: */
    if (WIFEXITED(status))
    {
        printf("Miner exited with status %d\n", WEXITSTATUS(status));
    }
    else
    {
        printf("Miner exited unexpectedly\n");
        exit(EXIT_FAILURE);
    }
    exit(EXIT_SUCCESS);
}

int main(int argc, char *argv[])
{
    int pid;
    int fd1_mon[2];
    int fd2_min[2];

    /* Comprobamos los argumentos: */
    if (argc != 4)
    {
        return EXIT_FAILURE;
    }

    long int target = atoi(argv[1]);
    int num_rounds = atoi(argv[2]);
    int num_threads = atoi(argv[3]);

    /* Fork! */
    pid = fork();

    if (pid < 0) /* Ha ocurrido un error */
    {
        perror("fork");
        exit(EXIT_FAILURE);
    }
    else if (pid == 0) /* Es hijo */
    {
        /* Creamos los pipes: */
        pipe(fd1_mon);
        pipe(fd2_min);
        pid = fork(); /* Fork! */
        if (pid < 0) /* Ha ocurrido un error */
        {
            perror("fork");
            exit(EXIT_FAILURE);
        }
        if (pid > 0) /* Es padre */
        {
            /* Llamada a minero: */
            llama_minero(fd2_min, fd1_mon, pid, target, num_rounds, num_threads);
        }
        else if (pid == 0) /* Es hijo */
        {
            /* Llamada a monitor: */
            llama_monitor(fd1_mon, fd2_min);
        }
    }
    else /* Es padre */
    {
        /* Llamada a padre: */
        llamada_padre(pid);
    }
    exit(EXIT_FAILURE);
}
```

## Apartado c)

En este apartado se implementa una función que comprueba el resultado obtenido por el minero en cada ronda y funciona de la siguiente manera:

En pimer lugar realiza la lectura de los datos pasados mediante la estructura de datos por medio de un pipe. Una vez recibidos el “target”y el “result”, se comprueba si se cumple “pow_has(result) = target”, en caso afirmativo devolverá al proceso monitor un “0” por medio de una pipe y en caso negativo le pasaría un “1” e imprime si la solución ha sido aceptada o rechazada.

```
/**
 * @brief Verifica las soluciones encontradas por el proceso minero.
 *
 *
 * @param fd_write int
 * @param fd_read int
 *
 * @return Returns EXIT_SUCCESS if there are not errors, EXIT_FAILURE otherwise.
 **/
int monitor(int fd_write, int fd_read)
{

    ThreadData thread_data;
    int ret, cero = 0, uno = 1;
    while (1)
    {
        /* Pasa la info del thread a thread_data: */
        ret = read(fd_read, &thread_data, sizeof(thread_data));

        if (ret == -1) /* Ha ocurrido un error */
        {
            printf("ERROR LEYENDO\n");
            perror("read");

            exit(EXIT_FAILURE);
        }
        else if (ret == 0) /* esta bien */
        {
            exit(EXIT_SUCCESS);
        }

        if (pow_hash(thread_data.result) == thread_data.target) /* Comprobamos si se cumple */
        {
            printf("Solution accepted: %08ld --> %08ld\n", thread_data.target, thread_data.result);

            /* Escribe la info del hilo en el pipe: */
            ret = write(fd_write, &cero, sizeof(int));

            if (ret == -1) /* Ha habido un error */
            {
                printf("ERROR ESCRIBIENDO 0\n");
                perror("write");

                exit(EXIT_FAILURE);
            }
        }
        else /* Si no se cumple */
        {
            printf("Solution rejected: %08ld !-> %08ld\n", thread_data.target, thread_data.result);

            /* Escribe la info del hilo en el pipe: */
            write(fd_write, &uno, sizeof(int));
```

```
            if (ret == -1) /* Ha habido un error */
            {
                printf("ERROR ESCRIBIENDO 1\n");
                perror("write");

                exit(EXIT_FAILURE);
            }
        }
    }
}
```