Sistemas Operativos

Universidad Autónoma De Madrid
Escuela Politécnica Superior
INGENIERIA INFORMATICA

Proyecto Final: Miner Rusher

DAVID TEÓFILO GARITAGOITIA ROMERO
DANIEL CERRATO SANCHEZ
ESAÚ ROMO GARCÍA

5/9/2021

Tabla de contenido

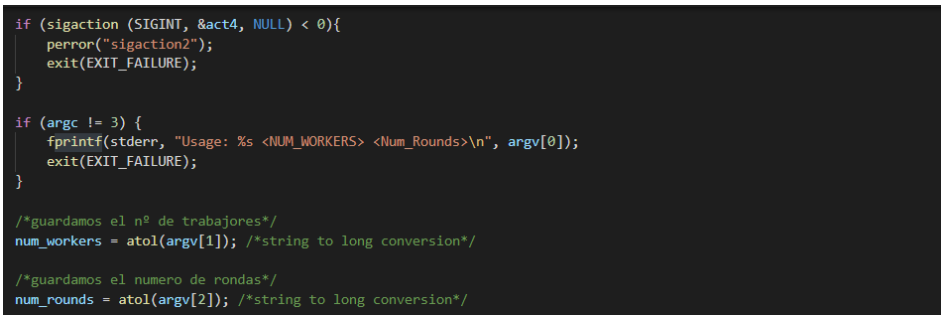
Ejercicio 1

El primer paso del proyecto fue la creación de los archivos ***miner.c*** y ***miner.h***. El primero de ellos contiene todo el código relacionado al minero, el cual ejecuta las rondas y obtiene una solución. El segundo de ellos es la cabecera, la cual contiene las estructuras de "***NetData*** y el ***Block***"

```c
#include <pthread.h>
#include <unistd.h>
#include <semaphore.h>

#define OK 0
#define MAX_MINERS 10
#define MAX_String 500
#define SHM_NAME_NET "/netdata"
#define SHM_NAME_BLOCK "/block "
#define SEM_NAME "/sem_winner "
#define SEM_HILOS "/sem_hilos"
#define MQ_NAME "/monitor"
/**
 *  En caso de que la solucion sea correcta, debera actualizar su cadena de bloques creando
 *  un nuevo bloque con la informaci´on compartida por el ganador en memoria compartida, en el
 *  segmento de nombre SHM_NAME_BLOCK:
*/
typedef struct _Block {
    int wallets [MAX_MINERS]; //registro de monedas conseguidas por cada mineto
    long int target; //número del que se quiere encontrar solución
    long int solution; //operando que da solución al target
    int id; //identificador de bloque
    int is_valid; //si considera como válida la respuesta
    struct _Block * next ; //puede no utilizarse
    struct _Block * prev ; //puede no utilizarse
    sem_t winner; //semaforo para usar al escribir la solucion del POW en memoria compartida
    int quorum;
    sem_t vote[MAX_MINERS];
    int mutex;
    sem_t semMutex;
}Block ;

typedef struct _NetData{
    pid_t miners_pid [MAX_MINERS]; //tabla con los identificadores de proceso de los mineros
    char voting_pool [MAX_MINERS]; //votos de cada minero
    int last_miner; //posición en la tabla del último minero activo
    int total_miners; //número total de mineros
    pid_t monitor_pid; //pid del monitor, se empleará uno
    pid_t last_winner; //ultimo minero ganador
}NetData;

long int simple_hash(long int number);
void print_blocks(Block * plast_block, int num_wallets);
```

Ilustración 1.- Estructuras declaradas en miner.h

Para el ***miner.c*** lo primero es declarar todas las variables (del tipo que sean) que servirán para realizar el programa, además de inicializar las máscaras de señales con sus respectivos manejadores que ayudarán a la red de mineros a comunicarse entre ellos mismo.

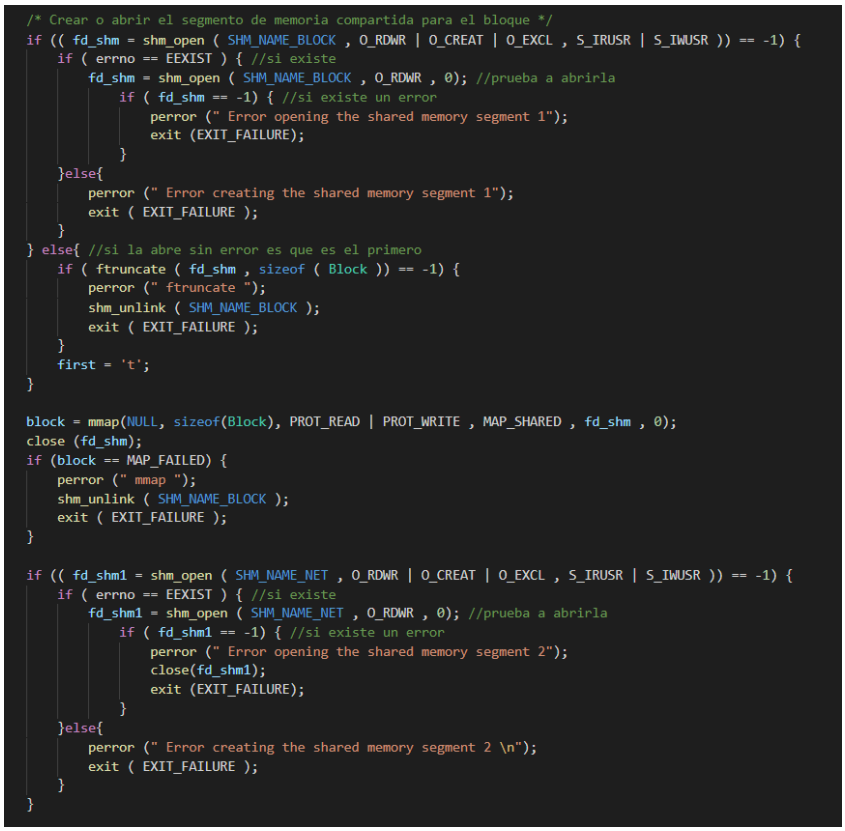


```c
if (sigaction (SIGINT, &act4, NULL) < 0){
    perror("sigaction2");
    exit(EXIT_FAILURE);
}

if (argc != 3) {
    fprintf(stderr, "Usage: %s <NUM_WORKERS> <Num_Rounds>\n", argv[0]);
    exit(EXIT_FAILURE);
}

/*guardamos el nº de trabajores*/
num_workers = atol(argv[1]); /*string to long conversion*/

/*guardamos el numero de rondas*/
num_rounds = atol(argv[2]); /*string to long conversion*/
```

Ilustración 2.- Recepción de los argumentos al llamar al minero por terminal.

Posterior a ello, se encuentra la recepción de los argumentos que se le pasan por terminal al programa, en el cual el primero de ellos corresponde al número de trabajadores que tendrán los mineros y el segundo argumento son el número de rondas a ejecutar. Estos valores se van a convertir de string a enteros.

Lo siguiente será la creación y mapeo de la memoria compartida para el ***Block*** y ***NetData*** con los que se podrá realizar tanto la comunicación y alojamiento de la información de los bloques realizados en la red por los mineros. Estos bloques contienen información importante de la red como puede ser: Wallets, coins, targets y soluciones, PID's, que debe estar visibles de todos los mineros y trabajadores.



```
/* Crear o abrir el segmento de memoria compartida para el bloque */
if (( fd_shm = shm_open ( SHM_NAME_BLOCK , O_RDWR | O_CREAT | O_EXCL , S_IRUSR | S_IWUSR )) == -1) {
    if ( errno == EEXIST ) { //si existe
        fd_shm = shm_open ( SHM_NAME_BLOCK , O_RDWR , 0); //prueba a abrirla
            if ( fd_shm == -1) { //si existe un error
                perror (" Error opening the shared memory segment 1");
                exit (EXIT_FAILURE);
            }
    }else{
        perror (" Error creating the shared memory segment 1");
        exit ( EXIT_FAILURE );
    }
} else{ //si la abre sin error es que es el primero
    if ( ftruncate ( fd_shm , sizeof ( Block )) == -1) {
        perror (" ftruncate ");
        shm_unlink ( SHM_NAME_BLOCK );
        exit ( EXIT_FAILURE );
    }
    first = 't';
}

block = mmap(NULL, sizeof(Block), PROT_READ | PROT_WRITE , MAP_SHARED , fd_shm , 0);
close (fd_shm);
if (block == MAP_FAILED) {
    perror (" mmap ");
    shm_unlink ( SHM_NAME_BLOCK );
    exit ( EXIT_FAILURE );
}

if (( fd_shm1 = shm_open ( SHM_NAME_NET , O_RDWR | O_CREAT | O_EXCL , S_IRUSR | S_IWUSR )) == -1) {
    if ( errno == EEXIST ) { //si existe
        fd_shm1 = shm_open ( SHM_NAME_NET , O_RDWR , 0); //prueba a abrirla
            if ( fd_shm1 == -1) { //si existe un error
                perror (" Error opening the shared memory segment 2");
                close(fd_shm1);
                exit (EXIT_FAILURE);
            }
    }else{
        perror (" Error creating the shared memory segment 2 \n");
        exit ( EXIT_FAILURE );
    }
}
```

Ilustración 3.- Fragmento de código de la creación de la memoria compartida con shm_open.

Una vez creada la memoria compartida, el primer minero en la red se encarga de esto y de realizar el mapeo de la memoria, después se procede a declarar los semáforos necesarios que sincronizarán el acceso de todos los trabajadores y mineros a poder leer y escribir en el bloque. Además, se pretende prevenir y eliminar la inanición y el interbloqueo.

```c
if ( first == 't' && (ftruncate ( fd_shm1 , sizeof ( NetData )) == -1)) {
    perror (" ftruncate ");
    shm_unlink ( SHM_NAME_NET );
    exit ( EXIT_FAILURE );
}

netdata = mmap(NULL, sizeof(NetData), PROT_READ | PROT_WRITE , MAP_SHARED , fd_shm1 , 0);
close (fd_shm1);
if (netdata == MAP_FAILED || netdata==NULL) {
    perror (" mmap ");
    munmap(block, sizeof(Block));
    shm_unlink ( SHM_NAME_BLOCK );
    shm_unlink ( SHM_NAME_NET);
    exit ( EXIT_FAILURE );
}

if(first == 't'){
    if (sem_init(&(block->winner),1,1) == -1){
        perror (" sem_init: winner");
        munmap(netdata, sizeof(NetData));
        munmap(block, sizeof(Block));
        shm_unlink ( SHM_NAME_BLOCK );
        shm_unlink ( SHM_NAME_NET);
        exit ( EXIT_FAILURE );
    }

    if (sem_init(&(block->semMutex),1,0) == -1){
        perror (" sem_init:semMutex");
        munmap(netdata, sizeof(NetData));
        munmap(block, sizeof(Block));
        shm_unlink ( SHM_NAME_BLOCK );
        shm_unlink ( SHM_NAME_NET);
        exit ( EXIT_FAILURE );
    }

    for(i = 0; i < MAX_MINERS; i++){
        if(sem_init(&(block->vote[i]),1,0) == -1){
            perror("sem_init: vote");
            sem_destroy(&(block->semMutex));
            sem_destroy(&(block->winner));
            munmap(netdata, sizeof(NetData));
            munmap(block, sizeof(Block));
            shm_unlink ( SHM_NAME_BLOCK );
            shm_unlink ( SHM_NAME_NET);
            exit ( EXIT_FAILURE );
```

Ilustración 4.- Fragmento de código de la inicialización y declaración de semáforos por el primer minero de la red.

El primer minero en la red también es el encargado de inicializar el bloque, algunas variables, algunos semáforos y de inicializar la estructura que tendrán los trabajadores.

```c
propio = (Block *)calloc(1, sizeof(Block));
if(propio == NULL){
    sem_unlink(SEM_NAME);
    munmap(netdata, sizeof(NetData));
    munmap(block, sizeof(Block));
    shm_unlink ( SHM_NAME_BLOCK );
    shm_unlink ( SHM_NAME_NET);
    exit ( EXIT_FAILURE );
}

if((ps = (Pthread_struct *)calloc(1, sizeof(Pthread_struct))) == NULL){
    free(propio);
    sem_unlink(SEM_NAME);
    munmap(netdata, sizeof(NetData));
    munmap(block, sizeof(Block));
    shm_unlink ( SHM_NAME_BLOCK );
    shm_unlink ( SHM_NAME_NET);
    exit ( EXIT_FAILURE );
}

if(first == 't'){ //si es el primero
    //inicializar block
    for(int i; i<MAX_MINERS; i++){
        block->wallets [i]= 0;
    }
    block->target = rand() % PRIME;  //número del que se quiere encontrar solución
    block->target = simple_hash(500000);  //número del que se quiere encontrar solución
    block->solution = -1; //operando que da solución al target
    block->id = 0;
    block->is_valid = -1;
    block->next = NULL;
    block->prev = NULL; //puede no utilizarse
    block->quorum = 0;
    block->mutex = 1;

    netdata->monitor_pid = 0; //pid del monitor, se empleará uno
    netdata->total_miners= 1; //número total de mineros
    netdata->last_winner = -1; //ultimo minero ganador
    netdata->last_miner=0; //posición en la tabla del último minero activo
    for(i=0; i<MAX_MINERS; i++)
        netdata->miners_pid[i] = -1;
    netdata->miners_pid[0] = getpid();
    miPos = 0;
}
```

Ilustración 5.- Fragmento de la inicialización del bloque por el primer minero.

Una vez hecho esto, se crean los hilos o trabajadores acorde a los indicados en los argumentos de entrada, los cuales serán hilos encargados de buscar la solución al problema.

```c
while(num_rounds--){

    for(i=0; i<netdata->total_miners; i++)
        printf("Minero %ld: %d\n", i, netdata->miners_pid[i]);

    printf("%d - rondas restantes: %ld\n", getpid(), num_rounds);
    if((workers = (pthread_t *)calloc(num_workers,sizeof(workers[0])))==NULL){ //creas el array de hilos
        free(ps);
        free(propio);
        munmap(netdata, sizeof(NetData));
        munmap(block, sizeof(Block));
        shm_unlink ( SHM_NAME_BLOCK );
        shm_unlink ( SHM_NAME_NET);
        exit(EXIT_FAILURE);
    }


    printf("Target del siguiente bloque: %ld\n", block->target);

    ps->id = getpid();
    ps->is_valid = 0;
    ps->target = block->target;
    ps->solucion = -1;
    ps->solucion_encontrada = 0;
    if(sem_init(&(ps->hilos), 0, 0) == -1){
        free(ps);
        free(propio);
        munmap(netdata, sizeof(NetData));
        munmap(block, sizeof(Block));
        shm_unlink ( SHM_NAME_BLOCK );
        shm_unlink ( SHM_NAME_NET);
        exit(EXIT_FAILURE);
    }

    sol = 0;
    for(i=0; i<num_workers; i++){
        if(pthread_create(&workers[i], NULL, trabajador, ps) != 0) //pones a los trabajadores

    }

    for(i=0; i<num_workers; i++){
        if(workers[i])
            pthread_join(workers[i], NULL);
    }

    free(workers);
```

Ilustración 6.- Fragmento de la creación de los trabajadores o hilos.

Una vez lanzados lo hilos, estos comienzan a buscar la solución, si alguno de ellos la encuentra, este se comunicará con su minero padre por medio del uso de señales*.* Cuando un hilo encuentra la solución, bloquea al resto de trabajadores en espera de la respuesta del padre ante la solución.

```c
/*worker*/
void *trabajador(void *s){
    long int miSol;
    while(sol < PRIME) {
        if(ps->solucion_encontrada){ /*si solución encontrada*/
            sem_wait(&ps->hilos); /*se paran y esperan a que el proceso padre los reactive*/
            sem_post(&ps->hilos); /*esperan a reactivarse y le envian a otro que este esperando que se active y así todo el roto*/
            if(ps->is_valid){
                pthread_exit(NULL);
            }
        }

        miSol = sol++;
        fprintf(stdout, "Searching... %6.2f%%\r", 100.0 * miSol / PRIME);
        if (ps->target == simple_hash(miSol) && !ps->is_valid) { //si encuentra la solución
            fprintf(stdout, "\nSolution: %ld\n", miSol);
            ps->solucion_encontrada = 1;
            ps->solucion = miSol; //pone el valor de la solución
            kill(ps->id, SIGCHLD); //comunica al padre que ha encontrado una solución
        }
    }
    fprintf(stderr, "\nSearch failed\n");
    exit(EXIT_FAILURE);
}
```

Ilustración 7.- Función a ejecutar por los trabajadores o hilos para encontrar la solución al problema y comunicarlo al minero.

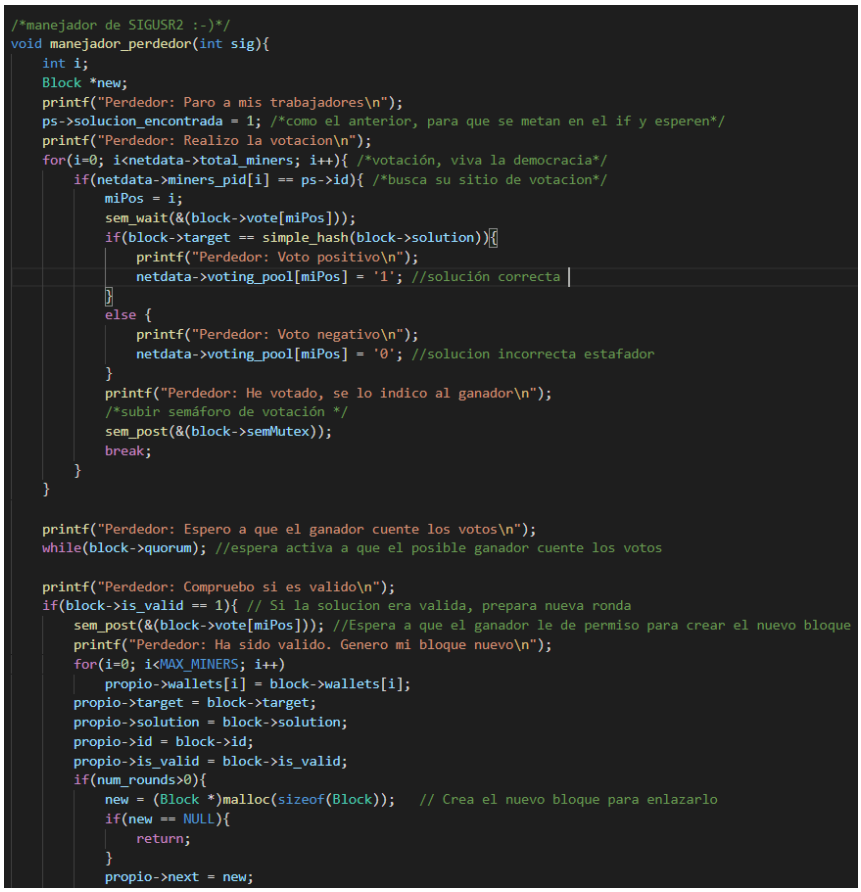
El Minero padre que pretende ser el ganador reacciona a la señal de hijo y manda la señal ***SIGNINT*** al último ganador que se encargará de actualizar la lista de mineros activos, al finalizar el manejador vuelve a poner el quorum a 0 para que de esta forma el posible ganador pueda salir de su espera activa, tras ello, actualizará el bloque de solución y mandará al resto de mineros la señal ***SIGUSR2***, a la cual responderán los mineros con el manejador detener, bloquean a sus trabajadores y votan si la solución les parece correcta o no para luego poner su semáforo de votación en activo, este semáforo permitirá al posible ganador esperar a la votación de todos los mineros activos, tras lo cual si el voto resulta favorable pondrá el bloque como válido, en caso contrario lo pondrá como no válido y reactivará sus trabajadores, el quorum volverá a ponerse a 0 lo que servirá al resto de mineros para salir de su espera activa y comprobar si el voto fue o no favorable, en caso de serlo crean un nuevo bloque y actualizan la lista enlazada en caso de no serlo reactivan a sus trabajadores.

```c
/*manejador del minero cuando un trabajador tiene una posible solución encontrada manejador sigchild*/
void manejador_ganador (int sig){ //, Pthread_struct *ps, Block *block, NetData *netdata
    int i, votos = 0, fd_shm;
    Block *new, *newPropio;

    printf("Ganador - Cierro la seccion critica\n");
    if(netdata->total_miners > 1){
        while(1){printf("Ganador - Bajo el semaforo mutex\n");
            sem_wait(&(block->winner));    /*baja el semáforo de ganador*/
            printf("Ganador - He pasado del semaforo\n");
            if(block->mutex == 1){      //metodo para evadir la posibilidad de interrupciones
                block->mutex = 0;
                break;                  //y su posterior salto de comando
            }
            if(ps->is_valid){
                printf("Ganador - Realmente perdi\n");
                return;
            }
        }
    }
    printf("Ganador - Preparo el recuento de activos\n");
    block->quorum = 1; /*sémaforo de espera activa*/
    if(netdata->last_winner == -1){
        netdata->last_winner = getpid();
    }
    printf("Ganador - Envio la señal al antiguo ganador\n");
    kill(netdata->last_winner, SIGCONT); /*envia un sigcont al ganador de la ronda anterior (el que actua como monitor*/
    printf("Ganador - Espero al antiguo ganador\n");
    while(block->quorum); /*bloque de espera activa hasta actualizar la lista de mineros activos*/
    block->quorum = 1; /*para que el resto de mineros después de votar esperen*/
    printf("Ganador - Preparo los votos: %d\n", netdata->total_miners);
    block->solution = ps->solucion; /*actualiza la solución*/
    for(i=0; i<netdata->total_miners; i++){ /*bucle por todos los mineros activos para comprobar su voto*/
        netdata->voting_pool[i] = '-'; //se inicializa a un valor diferente a ok o no ok :-)
        if(ps->id != netdata->miners_pid[i]){ /*comprobación de que no se auto envie*/
            printf("Ganador - Envio señal al perdedor\n");
            kill(netdata->miners_pid[i],SIGUSR2);
        }
        else
            miPos = i;
    }
    printf("Ganador - Espero a que voten\n");
    /* espera a la votacion */
    for(i=0; i!=netdata->total_miners; i++){
        if(netdata->miners_pid[i] != ps->id){
            sem_post(&(block->vote[i]));
```

Ilustración 8.- Fragmento del manejador del minero ganador que en caso de ganar actualiza el bloque con su solución.

```c
*/
void manejador_antiguo_ganador(int sig){
    pid_t activos[MAX_MINERS];
    int i, r, coins[MAX_MINERS];
    sem_t semaforos[MAX_MINERS];

    netdata->total_miners = 0;
    for(i = 0; i<MAX_MINERS; i++){
        if(netdata->miners_pid[i] != -1){
            if(getpid() != netdata->miners_pid[i]){
                r = kill(netdata->miners_pid[i], SIGUSR1);
                if(r == 0){
                    activos[netdata->total_miners] = netdata->miners_pid[i];
                    coins[netdata->total_miners] = block->wallets[i]; printf("Minero: %d tiene %d monedas\n", netdata->miners_pid[i], block->wallets[i]);
                    semaforos[netdata->total_miners] = block->vote[i];
                    netdata->total_miners++;
                }
            }
            else miPos = i;
        }
    }
    activos[netdata->total_miners] = getpid();
    coins[netdata->total_miners] = block->wallets[miPos];
    semaforos[netdata->total_miners] = block->vote[miPos];
    netdata->last_miner = netdata->total_miners;
    netdata->total_miners++;

    for(i = 0; i < netdata->total_miners; i++){
        netdata->miners_pid[i] = activos[i];
        block->wallets[i] = coins[i];
        block->vote[i] = semaforos[i];
    }
    for(i = netdata->total_miners; i<MAX_MINERS; i++){
        netdata->miners_pid[i] = -1;
        block->wallets[i] = 0;
        sem_destroy(&(block->vote[i]));
        sem_init(&(block->vote[i]), 1, 0);
    }

    for(i=0; i<netdata->total_miners; i++){
        printf("Minero %d: %d\n", i, netdata->miners_pid[i]);
    }

    block->quorum = 0; //ya he terminado de hacer las comprobaciones de procesos activos, el resto de procesos terminan la espera activa
}
```

Ilustración 9.- Fragmento de código del ganador antiguo que se encarga de comprobar los mineros activos.



```c
/*manejador de SIGUSR2 :-)*/
void manejador_perdedor(int sig){
    int i;
    Block *new;
    printf("Perdedor: Paro a mis trabajadores\n");
    ps->solucion_encontrada = 1; /*como el anterior, para que se metan en el if y esperen*/
    printf("Perdedor: Realizo la votacion\n");
    for(i=0; i<netdata->total_miners; i++){ /*votación, viva la democracia*/
        if(netdata->miners_pid[i] == ps->id){ /*busca su sitio de votacion*/
            miPos = i;
            sem_wait(&(block->vote[miPos]));
            if(block->target == simple_hash(block->solution)){
                printf("Perdedor: Voto positivo\n");
                netdata->voting_pool[miPos] = '1'; //solución correcta
            }
            else {
                printf("Perdedor: Voto negativo\n");
                netdata->voting_pool[miPos] = '0'; //solucion incorrecta estafador
            }
            printf("Perdedor: He votado, se lo indico al ganador\n");
            /*subir semáforo de votación */
            sem_post(&(block->semMutex));
            break;
        }
    }

    printf("Perdedor: Espero a que el ganador cuente los votos\n");
    while(block->quorum); //espera activa a que el posible ganador cuente los votos

    printf("Perdedor: Compruebo si es valido\n");
    if(block->is_valid == 1){ // Si la solucion era valida, prepara nueva ronda
        sem_post(&(block->vote[miPos])); //Espera a que el ganador le de permiso para crear el nuevo bloque
        printf("Perdedor: Ha sido valido. Genero mi bloque nuevo\n");
        for(i=0; i<MAX_MINERS; i++)
            propio->wallets[i] = block->wallets[i];
        propio->target = block->target;
        propio->solution = block->solution;
        propio->id = block->id;
        propio->is_valid = block->is_valid;
        if(num_rounds>0){
            new = (Block *)malloc(sizeof(Block));   // Crea el nuevo bloque para enlazarlo
            if(new == NULL){
                return;
            }
            propio->next = new;
```

Ilustración 10.- Fragmento del manejador de minero perdedor el cual se encarga de votar y comprobar que la solución propuesta sea válida.

Por último, el proceso ganador de la última ronda se encarga de cerrar todos los semáforos, la memoria compartida, de matar todos los procesos y terminar el programa.

```c
void manejador_acabar(int sig){

    print_blocks(propio, MAX_MINERS);

    ps->is_valid = 1;
    sem_post(&(ps->hilos));

    for(int i = MAX_MINERS-1; i >= 0; i--){
        if(netdata->miners_pid[i] != getpid()){
            netdata->last_miner = netdata->miners_pid[i];
        }
    }
    if(netdata->last_winner == getpid()){
        netdata->last_winner = -1;
    }

    for(int i = 0; i < MAX_MINERS; i++){
        if(netdata->miners_pid[i] == getpid()){
            netdata->miners_pid[i] = -1;
            netdata->total_miners--;
            netdata->voting_pool[i] = '-';
        }
    }

    free(workers);
    sem_destroy(&(ps->hilos));
    free(ps);
    free(propio);
    munmap(block, sizeof(Block));
    if(netdata->total_miners == 0){
        for(int i=0; i<MAX_MINERS; i++){
            sem_destroy(&(block->vote[i]));
        }
        sem_destroy(&(block->winner));
        sem_destroy(&(block->semMutex));
        munmap(netdata, sizeof(NetData));
        shm_unlink (SHM_NAME_NET);
        shm_unlink ( SHM_NAME_BLOCK );
    }
    else
        munmap(netdata, sizeof(NetData));

    exit(EXIT_SUCCESS);
}
```

Ilustración 11.- Fragmento para terminar el programa.

El bucle se repetirá tantas rondas como se haya especificado, de esta manera estaría implementado el minero, el cual es capaz de crear la red completa.

[FINAL DE DOCUMENTO]