MEMORIA PRÁCTICA 2

El objetivo de esta práctica es, partiendo de la implementación de la práctica anterior, que cada solución encontrada sea validada por una mayoría de los mineros que componen la red antes de ser aceptada. Esto es porque un las redes de mineros basados en Blockchain, es de gran importancia la validación de las soluciones obtenidas.

En el ejercicio 10 se nos pide implementar un sistema de votación multiproceso, esto es que en cada iteración un proceso se presenta como candidato, y el resto de los procesos determinen si aceptan o no su propuesta. De momento esta determinación se realizará de manera aleatoria. Esta sincronización se realizará mediante señales y semáforos.

Este ejercicio se divide en varias partes diferenciales, que son las siguientes:

- Arranque del sistema y proceso principal:

  El sistema debe ser ejecutado con dos parámetros de la siguiente manera:

  ```
  ./voting <N_PROCS> <N_SECS>
  ```

  voting: nombre del ejecutable.

  <N_PROGS>: número de procesos que participaran en la votación.

  <N_SECS>: número máximo de segundos de actividad del sistema.

  El proceso principal se encargará de crear el número de procesos votante que se hayan especificado y almacenará la información del sistema en un fichero.

  Una vez este listo el sistema, el proceso Principar enviará la señal SIGUSR1 a todos los procesos Votante. Cuando Principal reciba la señal de interrupción SIGINT, enviará a todos los procesos Votante la señal SIGTERM para que estos finalicen y liberen sus recursos.

  Cuando estos terminen, se liberarán los recursos del sistema y terminará su ejecución mostrando el mensaje “Finishing by signal”.

- Los procesos Votante:

  Como hemos visto en el apartado anterior, estos procesos al arrancar quedan en espera de que el proceso Principal les avise de que el sistema está listo.

  Una vez reciben la señal que les indica esto (SIGUSR1), leerán la información del sistema para poder enviar señales a todos los procesos implicados, y comenzará a iterar.

  En estas iteraciones los procesos competirán para convertirse en el proceso Candidato, esto lo consigue el primero que lo solicite a través de señales. El resto de procesos Votante quedarán en espera hasta que comience la votación, en la que elegirán si aceptan o no al candidato. Esta votación se realiza a través de un fichero compartido por todos los procesos.

  Una vez esté lista la votación, el proceso Candidato enviará la señal SIGUSR2 a los procesos Votante para que registren su voto. Después se esperará a que se hayan registrado todos los votos en el

fichero, alternando la comprobación del fichero con esperas no activas de 1ms.

Cuando los procesos Votante reciban la señal de votar, generarán un voto aleatorio y lo registrarán en el fichero. Una vez votado, entrarán en una espera no activa hasta la siguiente ronda. Una vez finalizada la votación, el proceso candidato mostrará los resultados con una cadena con el formato “Candidate %d ⇒ [Y Y N Y N] ⇒ Accepted” / “Candidate %d ⇒ [YNNNY] ⇒ Rejected”. El número indica el PID del proceso Y/N el sentido del voto de cada proceso, y la candidatura es aceptada en el caso de que el número de votos positivos sea mayor que el de los negativos. Tras esto se realizará una espera no activa de 250ms y arrancará una nueva ronda al enviar la señal SIGUSR1.

Los procesos Votante terminarán, liberando todos los recursos necesarios, al recibir la señal SIGTERM.

- Protección de zonas críticas:

  Es necesario garantizar que el sistema sea robusto para que no se pierda ninguna señal. Para ello, se deberán bloquear las señales y realizar las esperas no activas con sigsuspend.

- Temporización:

  En el caso de que el número máximo de segundos especificados por el sistema transcurra, el proceso Principal procederá a terminar el sistema enviando la señal SIGTERM a los procesos votantes que mostrarán el mensaje “FInished by alarm”.

Codigo (no hemos podido terminarlo y hemos dejado printfs para saber hasta donde va llegando):

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <time.h>

void crear_procesos(int n_proc);
void handler_sig1(int sig);
void handler_sigalarm(int sig);
int votante(int pid);
void serCandidato(int pid);
void solicitarVoto(int pid);
void votar(int pid);

int n_procs;
int n_secs;
FILE *f;
int hayCandidato = 0;

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

    if(argc != 3){
        printf("Error: ./voting numero de argumentos incorrecto\n");
    }
```

```
    f = fopen("voting.txt", "w");
    fclose(f);
    int n_proc = atoi(argv[1]);
    int n_secs = atoi(argv[2]);

    crear_procesos(n_proc);

    f = fopen("voting.txt", "r");
    while(!feof(f)){
        fscanf(f, "%d", &pid);
        kill(pid, SIGUSR1);
    }
    return 0;
}


void crear_procesos(int n_proc){
    int i;
    pid_t pid;

    struct sigaction sig1;
    sig1.sa_handler = handler_sig1;
    sigemptyset(&sig1.sa_mask);
    sig1.sa_flags = 0;

    struct sigaction sigalarm;
    sigalarm.sa_handler = handler_sigalarm;
    sigemptyset(&sigalarm.sa_mask);
    sigalarm.sa_flags = 0;

    sigaction(SIGUSR1, &sig1, NULL);

    for(i = 0; i < n_proc; i++){
        pid = fork();
        if(pid == 0){
            printf("Soy el proceso hijo %d y mi padre es %d\n", getpid(), getppid());
            sigaction(SIGUSR1, &sig1, NULL);
            printf("PREPAUSE\n");
            pause();
            printf("POSTPAUSE\n");
            votante(getpid());
            exit(0);
        }
        else if(pid > 0){
            printf("Soy el proceso padre %d y mi hijo es %d\n", getpid(), pid);
        }
        else{
            printf("Error al crear el proceso hijo %d\n", i);
        }
        f = fopen("voting.txt", "a");
        fprintf(f, "%d\n", pid);
        fclose(f);
    }
    sigaction(SIGALRM, &sigalarm, NULL);
    alarm(n_secs);


}


void handler_sig1(int sig){
    printf("Soy el proceso %d y he recibido la señal %d\n", getpid(), sig);

    if(sig == SIGUSR1){
```

```
        kill(getpid(), SIGCONT);
    }
}

void handler_sig2(int sig){
    int pid;
    printf("Soy el proceso %d y he recibido la señal %d\n", getpid(), sig);
    f = fopen("voting.txt", "r");
    for (int i = 0; i < n_procs; i++)
    {
        fscanf(f, "%d\n", &pid);
        kill(pid, SIGCONT);
    }
    fclose(f);
}
void handler_sigalarm(int sig){
    int pid;
    f = fopen("voting.txt", "r");
    for (int i = 0; i < n_procs; i++)
    {
        fscanf(f, "%d\n", &pid);
        kill(pid, SIGTERM);
    }
    for (int i = 0; i < n_procs; i++)
    {
        fscanf(f, "%d\n", &pid);
        waitpid(pid, NULL, 0);
    }
    fclose(f);

    printf("Finishing by alarm\n");
    exit(0);
}

int votante(int pid){
    printf("llamando a serCandidato\n");
    serCandidato(pid);
    return 0;
}

void serCandidato(int pid){
    if(hayCandidato == 1){
        printf("Soy el proceso %d y ya hay candidato activo\n", pid);
        pause();
        votar(pid);
    }
    printf("Soy el proceso %d y SOY EL CANDIDATO\n", getpid());
    hayCandidato = 1;
    solicitarVoto(pid);
}

void solicitarVoto(int pid){
    int pidCandidato;
    struct sigaction sig2;
    sig2.sa_handler = handler_sig2;
    sigaction(SIGUSR2, &sig2, NULL);
    printf("Soy el proceso %d y solicito voto\n", pid);
    sleep(1);
    f = fopen("voting.txt", "r");
    for(int i = 0; i < 10; i++){
        fscanf(f, "%d", &pidCandidato);
        if(pidCandidato != pid){
            printf("Soy el proceso %d y le envio la señal SIGUSR2 al proceso %d\n", getpid(), pidCandidato);
            kill(pidCandidato, SIGUSR2);
        }
```

```
        sleep(1);
    }
    hayCandidato = 0;
    printf("Soy el proceso %d y ya no soy candidato\n", pid);
    exit(0);
}

void votar(int pid){
    printf("Soy el proceso %d y voy a votar al proceso %d\n", getpid(), pid);
    //genera un numero aleatorio entre 0 y 1
    int voto = rand() % 2;
    f = fopen("votos.txt", "a");
    fprintf(f, "%d %d\n", getpid(), voto);
    fclose(f);

}
```