Sistemas Operativos

Práctica 2

Sistema de votación multiproceso

Angela Valderrama Ricaldi
Luis Felice Ruggiero
27/03/2023

Introducción

En esta segunda práctica se continúa con la implementación del proyecto final de la asignatura Sistemas Operativos, utilizando los conocimientos relacionados las señales y los semáforos que permiten la comunicación entre procesos y la sincronización entre ellos.

El ejercicio de codificación consiste en la implementación de un sistema de votación multiproceso en el cual, nuestro programa principal *voting* creará un número determinado de procesos hijos que competirán por convertirse en el proceso Candidato. Esto se regulará mediante el uso de un semáforo con nombre *candsem* que permitirá la distinción entre proceso Candidato y procesos Votante. La votación concurrente y el acceso al fichero donde se guardarán los votos, se controlarán con el semáforo *votsem.*

Las señales entran en acción cuando se desea que los procesos se comuniquen entre ellos. El proceso Principal comunicará a los procesos hijos que el sistema está listo con la señal SIGUSR1, el proceso Candidato comunicará a los procesos Votante que puede empezar la votación con la señal SIGUSR2 y por último el proceso Principal comunicará a los procesos hijos que deben terminar su ejecución con la señal SIGTERM.

Además, se ha implementado una alarma de un número determinado de segundos, que se indicarán al arrancar el programa, y que hará que el proceso Principal termine la ejecución si el tiempo indicado es excedido.

Para mantener una organización en el código, se han implementado dos módulos: Voting con el programa principal y Votante con la funcionalidad de la competición y los procesos Candidato y Votante.

Módulos

Voting

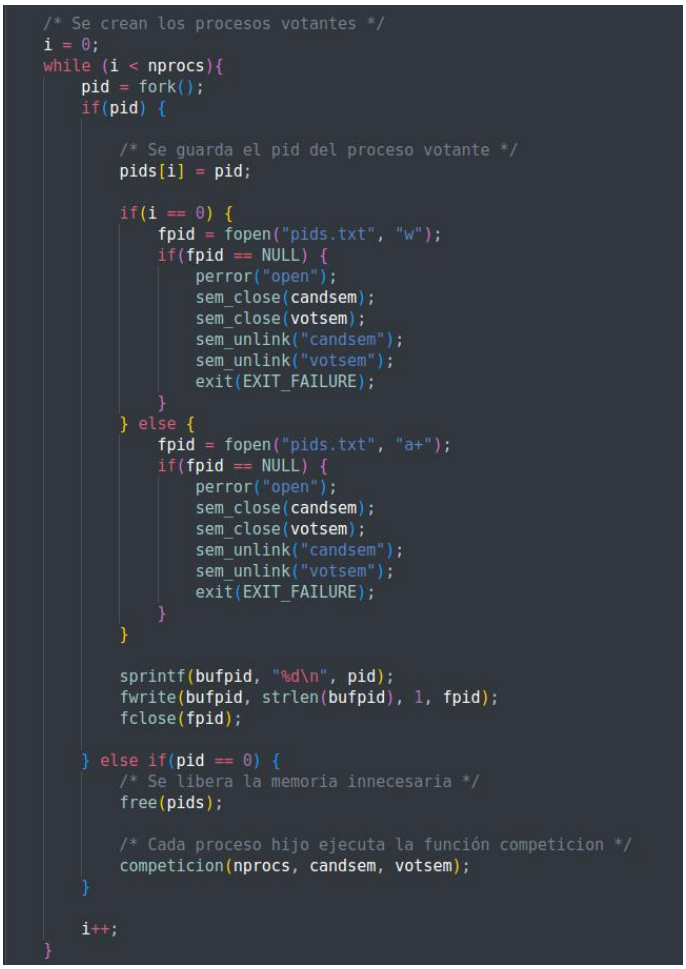
Este módulo es el principal del ejercicio de codificación. Aquí, se controlará la creación de los procesos Votante y el correcto funcionamiento de las señales capturadas en este proceso. La única función auxiliar de este módulo es el manejador de señales creado para las señales SIGINT y SIGALRM.

La función *void sighandler(int sig)* actualizará la variable global *got_signal* con el valor 1 cuando recibamos la señal SIGINT y con el valor 2 cuando recibamos la señal SIGALRM. Esta distinción se realiza para poder imprimir por pantalla el motivo de la finalización del programa.

El programa principal recibe como argumentos el número de procesos votantes que se van a crear y el número máximo de segundos que estará en funcionamiento el programa. De esta manera, el comando para ejecutar el programa, es el siguiente: *./voting <N_PROCS> <N_SECS>*

Después de recibir el número de procesos, se reserva memoria para guardar los PID de los procesos hijos. Además, se configura la captura de señales y la máscara de señales a utilizar. Se crean dos máscaras de señales: set y oldset. En la máscara set se bloquean las señales a capturar para que no interrumpan el flujo del programa y sólo se esperarán a ellas cuando se ejecute la función sigsupend(&oldset) que cambiará la máscara de set por la de oldset para poder recibir la señal sin problema.

Adicionalmente, se crearán dos semáforos con nombre inicializados a 1: *candsem,* que controlará la elección del proceso candidato y *votsem* que controlará el acceso concurrente al fichero de votos. De esta manera, se sincroniza la elección del proceso candidato (que consistirá en una suerte de condición de carrera) y la lectura y escritura del fichero donde se guardarán los votos.



```c
/* Se crean los procesos votantes */
i = 0;
while (i < nprocs){
    pid = fork();
    if(pid) {

        /* Se guarda el pid del proceso votante */
        pids[i] = pid;

        if(i == 0) {
            fpid = fopen("pids.txt", "w");
            if(fpid == NULL) {
                perror("open");
                sem_close(candsem);
                sem_close(votsem);
                sem_unlink("candsem");
                sem_unlink("votsem");
                exit(EXIT_FAILURE);
            }
        } else {
            fpid = fopen("pids.txt", "a+");
            if(fpid == NULL) {
                perror("open");
                sem_close(candsem);
                sem_close(votsem);
                sem_unlink("candsem");
                sem_unlink("votsem");
                exit(EXIT_FAILURE);
            }
        }

        sprintf(bufpid, "%d\n", pid);
        fwrite(bufpid, strlen(bufpid), 1, fpid);
        fclose(fpid);

    } else if(pid == 0) {
        /* Se libera la memoria innecesaria */
        free(pids);

        /* Cada proceso hijo ejecuta la función competicion */
        competicion(nprocs, candsem, votsem);
    }

    i++;
}
```

Figura 1. Creación de procesos hijos votantes

Una vez se haya configurado la captura de señales y los semáforos a utilizar, se procede a la creación de los procesos hijos votantes. El proceso padre guardará los PID de los procesos hijos en el fichero *pids.txt* y los procesos hijos liberarán la memoria innecesaria del array de PIDs y ejecutarán la función *competición()* del módulo Votante.

Cuando ya se hayan creado todos los procesos hijos determinados por la variable *nprocs,* el proceso principal enviará la señal SIGUSR1 a los procesos votantes para comunicarles que el sistema está listo. El proceso principal se quedará en espera, con la función *sigsuspend(&oldset)*, a recibir la señal SIGINT o SIGALRM. Si se recibe cualquiera de las dos señales, procederá a terminar los procesos votantes enviando la señal SIGTERM y recogiéndolos con la función *waitpid()*.

Finalmente, se liberarán todos los recursos utilizados por los semáforos y el array de PIDs.

Votante

En este módulo se implementa la funcionalidad básica de los procesos hijos votantes que ha creado el proceso principal.

*void competicion(int nprocs, sem_t *candsem, sem_t *votsem)*

Esta es la función principal que ejecutarán todos los procesos hijos en el que competirán para ser candidato. Nada más empezar, cada proceso votante reservará memoria para guardar los PID de los otros procesos votante para que en el caso de que sean Candidato, puedan enviar la señales necesarias.

Después, se configura mediante la función *señales()*, la captura de señales y la máscara a utilizar.

Para empezar, se esperará a recibir la señal SIGUSR1 de parte del proceso principal y así saber que el sistema está listo. Por ello, se procede a leer del fichero *pids.txt* los PIDs de los otros procesos votantes que se guardarán el array de PIDs.

Una vez se hayan guardado los PIDs, se empiezan las rondas de votación siempre y cuando la señal SIGUSR1 se haya recibido. Para controlar la elección del proceso Candidato, se hace uso de la función *semtrywait(candsem)* que permite que el proceso no se bloquee en el semáforo si su valor ya está a 0. De esta manera, si un proceso ya ha pasado por ese semáforo (es decir, es Candidato), los demás cuando intenten disminuir ese valor no se bloquearán y ejecutarán el código correspondiente.

```
/* Rondas de votación */
while(got_SIGUSR1) {

    got_SIGUSR1 = 0;

    /* decidir si soy candidato o no */
    if(sem_trywait(candsem) == 0) {
        candidato(nprocs, candsem, votsem);
    }else{
        votante(votsem);
    }

}
```

Figura 2. Ejecución de las rondas de votación

Si no se hacen uso de semáforos no se puede garantizar el correcto funcionamiento del programa, porque no habría sólo un proceso Candidato. Si sólo se usaran señales, la elección del Candidato podría ser factible pero el acceso a los recursos compartidos no.

El proceso Candidato ejecutará la función *candidato()* y los otros procesos Votante ejecutarán *votante().*

void señales()

En esta función se configura la captura de las señales SIGUSR1, SIGUSR2 y SIGTERM que tendrán como manejador la función *void sighandler(int sig)* activando las variables globales que actúan como flag *got_SIGUSR1* y *got_SIGUSR2* o, en el caso de SIGTERM, liberar los recursos y terminar el proceso.

Utilizaremos dos máscaras de señales: *set* en la que se añaden SIGUSR1 y SIGUSR2 y *termset* donde se encuentra SIGTERM. Para empezar, se bloqueará *set* y *termset* sólo durante el acceso a los ficheros para evitar que no se queden abiertos durante su lectura o escritura.

```
void señales(){

    /* bloquear señales */
    sigemptyset(&set);
    sigaddset(&set, SIGUSR1);
    sigaddset(&set, SIGUSR2);

    sigprocmask(SIG_BLOCK, &set, &oldset);

    /* bloquear señal SIGTERM */
    sigemptyset(&termset);
    sigaddset(&termset, SIGTERM);

    /* configurar la captura de señales */
    act.sa_handler = sighandler;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;

    if(sigaction(SIGUSR1, &act, NULL) < 0) {
        perror("sigaction");
        exit(EXIT_FAILURE);
    }

    if(sigaction(SIGUSR2, &act, NULL) < 0) {
        perror("sigaction");
        exit(EXIT_FAILURE);
    }

    if(sigaction(SIGTERM, &act, NULL) < 0) {
        perror("sigaction");
        exit(EXIT_FAILURE);
    }

    return;
}
```

Figura 3. Configuración de las señales

*void candidato(int nprocs, sem_t *candsem, sem_t *votsem)*

El proceso que actúa como Candidato, ejecutará esta rutina recibiendo el número de procesos participantes en la competición y lo semáforos a utilizar. Para empezar, se procede a bloquear la señal SIGTERM para poder crear el fichero *votos.txt* y evitar que se quede abierto. A continuación, una vez el fichero de votación se haya creado, se enviará la señal SIGUSR2 a los procesos que actúan como Votante para que comiencen la votación.

Mientras, el proceso Candidato estará comprobando que todos los votos están escritos en el fichero cada 1ms. Mediante el semáforo *votsem* accederá al recurso compartido y procederá a su lectura. Si se verifica que todos los procesos Votante han inscrito su voto, el Candidato realizará el recuento e informará del resultado imprimiéndolo por pantalla.

```
/* resetear variables */
yes = 0;
no = 0;

/* leer votos */
while(!feof(fp)) {
    fscanf(fp, "%d\n", &voto);
    if(voto == 0) {
        fprintf(stdout, "N ");
        no++;
    } else {
        fprintf(stdout, "Y ");
        yes++;
    }
}

if(yes > no) {
    fprintf(stdout, "] => Accepted\n");
} else {
    fprintf(stdout, "] => Rejected\n");
}
```

Figura 4. Comprobación de votos por el proceso Candidato

Después, se cierra el fichero, se levanta el semáforo *votsem* porque se termina el acceso al fichero y *candsem* ya que se va a elegir un nuevo candidato. A continuación, se envía la señal SIGUSR1 a los procesos Votante para comunicarles que empieza una nueva ronda y el proceso Candidato cambia su variable global *got_SIGUSR1* a 1 para presentarse de nuevo a la competición. Antes de empezar una nueva ronda, se esperan 250ms y comienza la nueva ronda.

*void votante(sem_t *votsem)*

Esta es la rutina que ejecutarán los procesos que actuarán como Votante. Primero, se quedarán en espera hasta que reciban la señal SIGUSR2 por parte del Candidato y una vez la reciban se procederá a la votación. Esta votación concurrente, se controlará mediante el semáforo *votsem* ya que se accede a un recurso compartido, en este caso el fichero *votos.txt* que guardará los votos de todos los procesos votantes. Además, se bloquea la señal SIGTERM durante la escritura del fichero para que no interrumpa el flujo y no deje el archivo abierto. Este voto será aleatorio, por eso se hace uso de la función *rand()* y se escribe en el fichero.

```
/* abrir fichero para escribir los votos */
fp = fopen("votos.txt", "a+");
if(fp == NULL) {
    perror("fopen");
    return;
}

/* decidir voto aleatoriamente entre 0 y 1 */
voto = rand() % 2;

/* escribir voto en el fichero */
fprintf(fp, "%d\n", voto);
fclose(fp);
sem_post(votsem);
```

Figura 5. Votación de los procesos Votante

Una vez la votación se haya hecho, su sube el valor al semáforo *votsem,* se desbloquea la señal SIGTERM y se procede a realizar la espera de la señal SIGUSR1 para dar comienzo a la próxima ronda.