Sistemas Operativos

Práctica 6. Mecanismos de sincronización de procesos en Linux y Windows (semáforos)

Prof. Jorge Cortés Galicia

Competencia.

El alumno comprende el funcionamiento de los mecanismos de sincronización entre procesos cooperativos utilizando los semáforos como árbitros de acceso para el desarrollo de aplicaciones cooperativas tanto en el sistema operativo Linux como Windows.

Desarrollo.

1. A través de la ayuda en línea que proporciona Linux, investigue el funcionamiento de las funciones: **semget(), semop().** Explique los argumentos, retorno de las funciones y las estructuras y uniones relacionadas con dichas funciones.
2. Capture, compile y ejecute el siguiente programa. Observe su funcionamiento y explique.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int main(void)
{
  int i,j;
  int pid;
  int semid;
  key_t llave = 1234;
  int semban = IPC_CREAT | 0666;
  int nsems = 1;
  int nsops;
  struct sembuf *sops = (struct sembuf *) malloc(2*sizeof(struct sembuf));
  printf("Iniciando semaforo...\n");
  if ((semid = semget(llave, nsems, semban)) == -1) {
            perror("semget: error al iniciar semaforo");
            exit(1);
     }
  else
            printf("Semaforo iniciado...\n");
  if ((pid = fork()) < 0) {
       perror("fork: error al crear proceso\n");
       exit(1);
   }
  if (pid == 0) {
    i = 0;
    while (i < 3) {
     nsops = 2;
```

```
    sops[0].sem_num = 0;
    sops[0].sem_op = 0;
    sops[0].sem_flg = SEM_UNDO;

    sops[1].sem_num = 0;
    sops[1].sem_op = 1;
    sops[1].sem_flg = SEM_UNDO | IPC_NOWAIT;
    printf("semop: hijo llamando a semop(%d, &sops, %d) con:", semid, nsops);
    for (j = 0; j < nsops; j++)  {
      printf("\n\tsops[%d].sem_num = %d, ", j, sops[j].sem_num);
      printf("sem_op = %d, ", sops[j].sem_op);
      printf("sem_flg = %#o\n", sops[j].sem_flg);
    }
    if ((j = semop(semid, sops, nsops)) == -1) {
      perror("semop: error en operacion del semaforo\n");
    }
    else {
      printf("\tsemop: regreso de semop() %d\n", j);
      printf("\n\nProceso hijo toma el control del semaforo: %d/3 veces\n", i+1);
      sleep(5);
      nsops = 1;
      sops[0].sem_num = 0;
      sops[0].sem_op = -1;
      sops[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
      if ((j = semop(semid, sops, nsops)) == -1) {
              perror("semop: error en operacion del semaforo\n");
      }
      else
              printf("Proceso hijo regresa el control del semaforo: %d/3 veces\n", i+1);
      sleep(5);
    }
    ++i;
   }
}
else {
     i = 0;
     while (i  < 3) {
       nsops = 2;
       sops[0].sem_num = 0;
       sops[0].sem_op = 0;
       sops[0].sem_flg = SEM_UNDO;

       sops[1].sem_num = 0;
       sops[1].sem_op = 1;
       sops[1].sem_flg = SEM_UNDO | IPC_NOWAIT;
       printf("\nsemop: Padre llamando semop(%d, &sops, %d) con:", semid, nsops);
       for (j = 0; j < nsops; j++) {
               printf("\n\tsops[%d].sem_num = %d, ", j, sops[j].sem_num);
               printf("sem_op = %d, ", sops[j].sem_op);
               printf("sem_flg = %#o\n", sops[j].sem_flg);
       }
       if ((j = semop(semid, sops, nsops)) == -1) {
                perror("semop: error en operacion del semaforo\n");
       }
```

```
      else {
            printf("semop: regreso de semop() %d\n", j);
            printf("Proceso padre toma el control del semaforo: %d/3 veces\n", i+1);
            sleep(5);
            nsops = 1;
            sops[0].sem_num = 0;
            sops[0].sem_op = -1;
            sops[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
            if ((j = semop(semid, sops, nsops)) == -1) {
               perror("semop: error en semop()\n");
             }
             else
               printf("Proceso padre regresa el control del semaforo: %d/3 veces\n", i+1);
            sleep(5);
       }
      ++i;
    }
  }
}
```

3. Capture, compile y ejecute los siguientes programas. Observe su funcionamiento. Ejecute de la siguiente forma: C:\>nombre_programa_padre nombre_programa_hijo

```
#include <windows.h>                        /*Programa padre*/
#include <stdio.h>
int main(int argc, char *argv[])
{
   STARTUPINFO si;                    /* Estructura de información inicial para Windows */
   PROCESS_INFORMATION pi;       /* Estructura de información del adm. de procesos */
   HANDLE hSemaforo;
   int i=1;
   ZeroMemory(&si, sizeof(si));
   si.cb = sizeof(si);
   ZeroMemory(&pi, sizeof(pi));
   if(argc!=2)
   {
      printf("Usar: %s   Nombre_programa_hijo\n", argv[0]);
      return;
   }

   // Creación del semáforo
   if((hSemaforo = CreateSemaphore(NULL, 1, 1, "Semaforo")) == NULL)
   {
      printf("Falla al invocar CreateSemaphore: %d\n", GetLastError());
      return -1;
   }

   // Creación proceso hijo
   if(!CreateProcess(NULL, argv[1], NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
   {
      printf("Falla al invocar CreateProcess: %d\n", GetLastError() );
      return -1;
   }
```

```
while(i<4)
   {
   // Prueba del semáforo
   WaitForSingleObject(hSemaforo, INFINITE);

   //Sección crítica
   printf("Soy el padre entrando %i de 3 veces al semaforo\n",i);
   Sleep(5000);

   //Liberación el semáforo
   if (!ReleaseSemaphore(hSemaforo, 1, NULL) )
   {
      printf("Falla al invocar ReleaseSemaphore: %d\n", GetLastError());
   }
   printf("Soy el padre liberando %i de 3 veces al semaforo\n",i);
   Sleep(5000);

   i++;
   }

 // Terminación controlada del proceso e hilo asociado de ejecución
 CloseHandle(pi.hProcess);
 CloseHandle(pi.hThread);
}
```

```
#include <windows.h>                    /*Programa hijo*/
#include <stdio.h>
int main()
{
   HANDLE hSemaforo;
   int i=1;

   // Apertura del semáforo
   if((hSemaforo = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, "Semaforo")) ==
NULL)
   {
      printf("Falla al invocar OpenSemaphore: %d\n", GetLastError());
      return -1;
   }

  while(i<4)
   {
   // Prueba del semáforo
   WaitForSingleObject(hSemaforo, INFINITE);

   //Sección crítica
   printf("Soy el hijo entrando %i de 3 veces al semaforo\n",i);
   Sleep(5000);

   //Liberación el semáforo
   if (!ReleaseSemaphore(hSemaforo, 1, NULL) )
   {
      printf("Falla al invocar ReleaseSemaphore: %d\n", GetLastError());
   }
   printf("Soy el hijo liberando %i de 3 veces al semaforo\n",i);
   Sleep(5000);

   i++;
   }
}
```

4. Programe la misma aplicación del punto 7 de la práctica 5 (tanto para Linux como para Windows), utilizando como máximo tres regiones de memoria compartida de 400 bytes cada una para almacenar todas las matrices requeridas por la aplicación. Utilice como mecanismo de sincronización los semáforos revisados en esta práctica tanto para la escritura y como para la lectura de las memorias compartidas. Úselos en los lugares donde haya necesidad de sincronizar el acceso a memoria compartida.