Sistemas Operativos

Práctica 7. Comunicación inter procesos (IPC) en Linux y Windows

Prof. Jorge Cortés Galicia

Competencias.

El alumno comprende el funcionamiento de las tuberías (pipes) sin nombre y de la memoria compartida como mecanismos de comunicación entre procesos tanto en el sistema operativo Linux como Windows para el desarrollo de aplicaciones concurrentes con soporte de comunicación.

Desarrollo.

1. A través de la ayuda en línea que proporciona Linux, investigue el funcionamiento de las llamadas al sistema: **pipe(), shmget(), shmat().** Explique los argumentos y retorno de la función.
2. Capture, compile y ejecute el siguiente programa. Observe su funcionamiento.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#define VALOR 1
int main(void)
{
        int desc_arch[2];
        char bufer[100];
        if(pipe(desc_arch) != 0)
                exit(1);
        if(fork() == 0)
        {
                while(VALOR)
                {
                        read(desc_arch[0],bufer,sizeof(bufer));
                        printf("Se recibió: %s\n", bufer);
                }
        }
        while(VALOR)
        {
                printf("Escriba un mensaje: ");
                gets(bufer);
                write(desc_arch[1],bufer,strlen(bufer)+1);
        }
}
```

3. A través del sitio MSDN, investigue el funcionamiento de las llamadas al sistema: **CreatePipe(), GetStartupInfo(), GetStdHandle(), WaitForSingleObject().** Explique los argumentos y retorno de la función.
4. 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>
#include <stdio.h>
#include <string.h>

int main (int argc, char *argv[])
{
  char mensaje[]="Tuberias en Windows";
  DWORD escritos;
  HANDLE hLecturaPipe, hEscrituraPipe;
  PROCESS_INFORMATION piHijo;
  STARTUPINFO siHijo;
  SECURITY_ATTRIBUTES pipeSeg =
        {sizeof(SECURITY_ATTRIBUTES), NULL, TRUE};
  /* Obtención de información para la inicialización del proceso hijo */
  GetStartupInfo (&siHijo);
  /* Creación de la tubería sin nombre */
  CreatePipe (&hLecturaPipe, &hEscrituraPipe, &pipeSeg, 0);
  /* Escritura en la tubería sin nombre */
  WriteFile(hEscrituraPipe, mensaje, strlen(mensaje)+1, &escritos, NULL);

  siHijo.hStdInput  = hLecturaPipe;
  siHijo.hStdError  = GetStdHandle (STD_ERROR_HANDLE);
  siHijo.hStdOutput = GetStdHandle (STD_OUTPUT_HANDLE);
  siHijo.dwFlags = STARTF_USESTDHANDLES;
  CreateProcess (NULL, argv[1], NULL, NULL,
       TRUE,  /* Hereda el proceso hijo los manejadores de la tubería del padre */
       0, NULL, NULL, &siHijo, &piHijo);

  WaitForSingleObject (piHijo.hProcess, INFINITE);
  printf("Mensaje recibido en el proceso hijo, termina el proceso padre\n");
  CloseHandle(hLecturaPipe);
  CloseHandle(hEscrituraPipe);
  CloseHandle(piHijo.hThread);
  CloseHandle(piHijo.hProcess);
  return 0;
}
```

```
/* Programa hijo.c */

#include <windows.h>
#include <stdio.h>
int main ()
{
  char mensaje[20];
  DWORD leidos;
  HANDLE hStdIn = GetStdHandle(STD_INPUT_HANDLE);

  /* Lectura desde la tubería sin nombre */
  ReadFile(hStdIn, mensaje, sizeof(mensaje), &leidos, NULL);
  printf("Mensaje recibido del proceso padre: %s\n", mensaje);
  CloseHandle(hStdIn);
  printf("Termina el proceso hijo, continua el proceso padre\n");
  return 0;
}
```

5. Programe una aplicación que cree un proceso hijo a partir de un proceso padre, el proceso padre enviará al proceso hijo, a través de una tubería, dos matrices de 10 x 10 a multiplicar por parte del hijo, mientras tanto el proceso hijo creará un hijo de él, al cual enviará dos matrices de 10 x 10 a sumar en el proceso hijo creado, nuevamente el envío de estos valores será a través de una tubería. Una vez calculado el resultado de la suma, el proceso hijo del hijo devolverá la matriz resultante a su abuelo (vía tubería). A su vez, el proceso hijo devolverá la matriz resultante de la multiplicación que realizó a su padre. Finalmente, el proceso padre obtendrá la matriz inversa de cada una de las matrices recibidas y el resultado lo guardará en un archivo para cada matriz inversa obtenida.
   Programe esta aplicación tanto para Linux como para Windows utilizando las tuberías de cada sistema operativo.
6. Capture, compile y ejecute los siguientes programas para Linux. Observe su funcionamiento.

```
#include <sys/types.h>              /* Cliente de la memoria compartida */
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#define TAM_MEM     27              /*Tamaño de la memoria compartida en bytes */
int main()
{
   int shmid;
   key_t llave;
   char *shm, *s;
   llave = 5678;
   if ((shmid = shmget(llave, TAM_MEM, 0666)) < 0) {
      perror("Error al obtener memoria compartida: shmget");
      exit(-1);
   }
   if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
      perror("("Error al enlazar la memoria compartida: shmat");
      exit(-1);
   }
   for (s = shm; *s != '\0'; s++)
      putchar(*s);
   putchar('\n');
   *shm = '*';
   exit(0);
}
```

```
#include <sys/types.h>                /* Servidor de la memoria compartida */
#include <sys/ipc.h>                  /* (ejecutar el servidor antes de ejecutar el cliente)*/
#include <sys/shm.h>
#include <stdio.h>
#define TAM_MEM     27                /*Tamaño de la memoria compartida en bytes */
int main()
{
   char c;
   int shmid;
   key_t llave;
   char *shm, *s;
   llave = 5678;
   if ((shmid = shmget(llave, TAM_MEM, IPC_CREAT | 0666)) < 0) {
      perror("Error al obtener memoria compartida: shmget");
      exit(-1);
   }
   if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
      perror("Error al enlazar la memoria compartida: shmat");
      exit(-1);
   }
   s = shm;
   for (c = 'a'; c <= 'z'; c++)
      *s++ = c;
   *s = '\0';
   while (*shm != '*')
      sleep(1);
   exit(0);
}
```

7. Capture, compile y ejecute los siguientes programas para Windows. Observe su funcionamiento.

```
#include <windows.h>              /* Cliente de la memoria compartida */
#include <stdio.h>
#define TAM_MEM     27        /*Tamaño de la memoria compartida en bytes */
int main(void)
{
  HANDLE hMemCom;
  char *idMemCompartida = "MemoriaCompatida";
  char *apDatos, *apTrabajo, c;
  if((hMemCom =OpenFileMapping(
            FILE_MAP_ALL_ACCESS,   // acceso lectura/escritura de la memoria compartida
            FALSE,                 // no se hereda el nombre
            idMemCompartida)       // identificador de la memoria compartida
            ) == NULL)
  {
    printf("No se accedio a la memoria compartida: (%i)\n", GetLastError());
    exit(-1);
  }
  if((apDatos=(char *)MapViewOfFile(hMemCom,      // Manejador del mapeo
                FILE_MAP_ALL_ACCESS,            // Permiso de lectura/escritura en la memoria
                0,
                0,
                TAM_MEM)) == NULL)
  {
    printf("No se enlazo la memoria compartida: (%i)\n", GetLastError());
    CloseHandle(hMemCom);
    exit(-1);
  }
```

```
  for (apTrabajo = apDatos; *apTrabajo != '\0'; apTrabajo++)
      putchar(*apTrabajo);
   putchar('\n');
   *apDatos = '*';
  UnmapViewOfFile(apDatos);
  CloseHandle(hMemCom);
  exit(0);
}
```

```
#include <windows.h>                /* Servidor de la memoria compartida */
#include <stdio.h>                  /* (ejecutar el servidor antes de ejecutar el cliente)*/
#define TAM_MEM     27              /*Tamaño de la memoria compartida en bytes */
int main(void)
{
  HANDLE hMemCom;
  char *idMemCompartida = "MemoriaCompatida";
  char *apDatos, *apTrabajo, c;
  if((hMemCom =CreateFileMapping(
            INVALID_HANDLE_VALUE,   // usa memoria compartida
            NULL,                    // seguridad por default
            PAGE_READWRITE,          // acceso lectura/escritura a la memoria
            0,                       // tamaño maxixmo parte alta de un DWORD
            TAM_MEM,                 // tamaño maxixmo parte baja de un DWORD
            idMemCompartida)         // identificador de la memoria compartida
            ) == NULL)
  {
    printf("No se creo la memoria compartida: (%i)\n", GetLastError());
    exit(-1);
  }
  if((apDatos=(char *)MapViewOfFile(hMemCom,      // Manejador del mapeo
                FILE_MAP_ALL_ACCESS,              // Permiso de lectura/escritura en la memoria
                0,
                0,
                TAM_MEM)) == NULL)
  {
    printf("No se enlazo la memoria compartida: (%i)\n", GetLastError());
    CloseHandle(hMemCom);
    exit(-1);
  }
  apTrabajo = apDatos;
  for (c = 'a'; c <= 'z'; c++)
      *apTrabajo++ = c;
   *apTrabajo = '\0';
   while (*apDatos != '*')
      sleep(1);
  UnmapViewOfFile(apDatos);
  CloseHandle(hMemCom);
  exit(0);
}
```

8. Programe nuevamente la aplicación del punto 5 utilizando en esta ocasión memoria compartida en lugar de tuberías (utilice tantas memorias compartidas como requiera). Programe esta aplicación tanto para Linux como para Windows utilizando la memoria compartida de cada sistema operativo.
9. Programe la misma aplicación del punto 5 de la práctica 5 en su sección de Linux, pero en esta ocasión cada proceso que realiza una operación de matrices, comunicará con un IPC el resultado de la operación al proceso que recolectará los resultados. Realice la versión del programa usando tuberías (en Linux y Windows), y la versión usando memoria compartida (en Linux y Windows).

Utilice tantas tuberías como memorias compartidas requiera para comunicar correctamente a los procesos.