INGENIERÍA UNIVERSIDAD DE LOS ANDES MÉRIDA VENEZUELA | DEPARTAMENTO DE COMPUTACIÓN ESCUELA DE SISTEMAS

Laboratorio de Sistemas Operativos
Semestre A-2017
Práctica Cuatro
Laboratorio

Prof. Rodolfo Sumoza
Prep. Alvaro Araujo
Prep. Luis Sanchez

1. Comunicación interproceso IPC.

En los sistemas operativos es de gran importancia la comunicación entre los procesos, existen procesos que son de naturaleza cooperante lo cual trae beneficios como: la rapidez de cómputo, la modularidad, el poder compartir información, entre otros. Para que un proceso pueda cooperar con otros procesos este requiere de un canal de comunicación. A continuación se describe el funcionamiento de las principales formas de comunicación una cola de mensajes y un segmento de memoria compartida.

2. Memoria Compartida.

La comunicación a través de memoria compartida como su nombre lo indica requiere de que los procesos que se van a comunicar establezcan una región de memoria compartida. Habitualmente, el sistema operativo intenta evitar que un proceso acceda a la memoria de otro proceso. La memoria compartida requiere que dos o mas procesos acuerden eliminar esta restricción.

2.1. Elección de la llave para el segmento de memoria compartida.

La obtención de una dirección para el segmento de memoria compartida se puede lograr básicamente de dos formas:

- Definiendo una dirección lógica hexadecimal cualquiera. Si esta dirección ya se encuentra ocupada el sistema operativo asignará otra dirección lógica, a menos que dicha dirección este ocupada por otro segmento de memoria compartida y el sistema interprete que se está refiriendo al segmento de memoria compartida ya creado.
- Con la función `ftok()`. Esta función convierte la ruta de un archivo y su identificador de proyecto en una clave IPC, permitiendo identificar un segmento de memoria compartida, los procesos que requieran conexión con el segmento deberán proveer la misma ruta y el mismo identificador de proyecto.

```
key_t ftok(const char *pathname, int proj_id);
```

También se debe crear una estructura con los campos que se desea contenga el segmento de memoria compartida.

2.2. Creación o búsqueda del segmento de memoria compartida.

Luego de haber asignado la dirección que tendrá el nuevo segmento de memoria compartida, se puede crear el segmento en cuestión con la siguiente función:

```
key_t shmget(key_t key, size_t size, int shmflg);
```

Para usar esta función con el fin de crear un segmento de memoria, se debe pasar como parámetro `shmflg` la permisologia en formato octal y la bandera IPC_CREAT la cual indica si se debe crear la memoria en caso de que no exista. En caso de solo querer obtener el identificador de un determinado segmento de memoria existente, solo es necesario enviar como parámetro `shmflg` la permisologia en formato octal. Existen una variedad de configuraciones para el parámetro `shmflg`, sin embargo estas dos serán suficientes por los momentos.

2.3. Enlace al segmento de memoria compartida.

Luego de conocer el identificador del segmento de memoria, un proceso se puede enlazar a dicho segmento haciendo uso de la siguiente función:

```
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

Esta función devuelve un puntero al segmento de memoria especificado.

2.4. Desvinculación y eliminación del segmento de memoria compartida.

Cuando ya no es necesario el segmento de memoria, este debe ser liberado, hay dos funciones básicas para liberar este espacio de memoria, la primera desvincula a un proceso de dicho segmento de memoria compartida, la sintaxis de ésta función es la siguiente:

```
int shmdt(const void *shmaddr);
```

Con la segunda función se puede liberar el espacio ocupado por el segmento de memoria compartida, la naturaleza de esta función es aplicar operaciones de control sobre un segmento de memoria, si se quiere eliminar el segmento se debe enviar como parámetro `cmd` la bandera IPC_RMID, en este caso se asigna el valor NULL al parámetro `shmid_ds`. La sintaxis de esta función es la siguiente:

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

Es muy importante liberar la memoria compartida ya que la culminación de los procesos vinculados a dicho segmento no significa la liberación automática de ese espacio de memoria.

2.5. Implementación de un segmento de memoria compartida.

En el siguiente ejemplo se observan dos programas, uno de ellos llamado monitor, al ejecutarse crea un segmento de memoria compartida y monitorea la actividad sobre un grupo de procesos asociados. El segundo programa, llamado operador, al ejecutarse realiza un conteo desde 1 hasta un numero aleatorio contenido en el rango [10-100]. A continuación se muestra el código fuente de ambos programas:

```
1  //monitor
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <unistd.h>
5  #include <sys/types.h>
6  #include <sys/shm.h>
7  #include <signal.h>
8  #include "memsh.h"
9
10 void shmem_init(shmem_data *);
11 void show_mon(shmem_data *);
12 void exit_signal(int);
13
14 int main()
15 {
```

```
16    system("clear");
17    printf("\n_____________________  Monitor De
          Procesos _____________________\n\n");
18
19    key_t id_shmem=ftok(ROUTE, ID);
20    void *pto_shmem;
21    shmem_data *pto_inf;
22    int i=0, shmem;
23
24    signal(2, exit_signal);
25
26
27    //Creacion del segmento de memoria compartida
28    if((shmem=shmget(id_shmem,sizeof(shmem_data),
          IPC_CREAT|0666))<0)
29    {
30      perror("shmget");
31      exit(EXIT_FAILURE);
32    }
33
34    //Vinculacion al segmento
35    if ((pto_shmem=shmat(shmem,NULL,0))==(char *)-1)
36    {
37      perror("shmat");
38      exit(EXIT_FAILURE);
39    }
40
41    //Inicializacion
42    pto_inf = (shmem_data *) pto_shmem;
43    shmem_init(pto_inf);
44
45    while(1)
46    {
47      show_mon(pto_inf);
48      usleep(100000);
49    }
50
51    return(0);
52  }
53
54
55  void shmem_init(shmem_data *pto_inf)
56  {
57    int i=0;
58    pto_inf->pid_mon = getpid();
59    for(i; i<10; i++)
```

```
60    {
61      pto_inf->array_p[i].pid = 0;
62          pto_inf->array_p[i].numero = 0;
63          pto_inf->array_p[i].termino = 0;
64    }
65  }
66
67  void show_mon(shmem_data *pto_inf)
68  {
69    int i=0;
70    system("clear");
71    printf("\n_________________________  Monitor De
        Procesos %d _________________________\n\n",pto_inf
        ->pid_mon);
72    printf("\t PID\tNUMERO\tTERMINO\n");
73    printf("\t-------------------------\n");
74    for(i; i<10; i++)
75    {
76      if(pto_inf->array_p[i].pid != 0)
77      {
78        printf(" \t%d\t%d\t", pto_inf->array_p[i].pid,
            pto_inf->array_p[i].numero);
79        fflush(stdout);
80        if(pto_inf->array_p[i].termino == 0)
81          printf("NO \n");
82        else
83          printf("YES\n");
84
85        fflush(stdout);
86      }
87    }
88  }
89
90  void exit_signal(int num_signal)
91  {
92    int i = 0, shmem;
93    key_t id_shmem = ftok(ROUTE, ID);
94
95    if((shmem = shmget(id_shmem,sizeof(shmem_data),0666)
        ) < 0)
96    {
97      perror("shmget");
98          exit(EXIT_FAILURE);
99    }
100
101   if (shmctl(shmem, IPC_RMID, 0) < 0)
```

```
  {
    perror("shmctl(IPC_RMID)");
        exit(EXIT_FAILURE);
  }

  system("clear");
  printf("Hasta luego!\n");
  exit(EXIT_SUCCESS);
}
```

```
//operador
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <signal.h>
#include "memsh.h"

int shmem_init(shmem_data *);

int main()
{
  system("clear");
  printf("\n--> Proceso %d \n\n",getpid());
  srand(getpid());

  key_t id_shmem = ftok(ROUTE, ID);
  void *pto_shmem;
  shmem_data *pto_inf;
  int i = 0, shmem, pos, repeticion;


  //Busqueda del segmento de memoria compartida
  if((shmem=shmget(id_shmem,sizeof(shmem_data),0666))
     < 0)
  {
        perror("shmget");
    exit(EXIT_FAILURE);
  }

  //Vinculacion al segmento
  if((pto_shmem=shmat(shmem, NULL, 0))==(char *)-1)
  {
    perror("shmat");
        exit(EXIT_FAILURE);
```

```
36    }
37
38    pto_inf = (shmem_data *) pto_shmem;
39    pos = shmem_init(pto_inf);
40
41    if(pos == -1)
42    {
43      if(shmdt(pto_shmem) == -1)
44          {
45            perror("shmdt");
46            exit(EXIT_FAILURE);
47          }
48
49      printf("\tMonitor sin espacio!!!\n\n");
50      exit(EXIT_SUCCESS);
51    }
52
53    repeticion = rand()%(100-10+1)+10;
54    for(i=0; i<repeticion; i++)
55    {
56      pto_inf->array_p[pos].numero++;
57      printf("Numero: %d\n",i);
58      usleep(500000);
59    }
60
61    pto_inf->array_p[pos].termino = 1;
62    if(shmdt(pto_shmem) == -1)
63    {
64      perror("shmdt");
65          exit(EXIT_FAILURE);
66    }
67
68    return(0);
69  }
70
71  int shmem_init(shmem_data *pto_inf)
72  {
73    int i=0;
74    pto_inf->pid_mon = getpid();
75    for(i; i<10; i++)
76      if(pto_inf->array_p[i].pid == 0)
77          {
78            pto_inf->array_p[i].pid = getpid();
79            return i;
80          }
81    return -1;
```

```
}
```

```
//memsh.h
#define ID 999
#define ROUTE "/bin/lspci"

typedef struct
{
  pid_t pid;
  long long numero;
  unsigned char termino;
}inf_p;

typedef struct{
  pid_t pid_mon;
        inf_p array_p[10];
}shmem_data;
```

3. Cola de mensajes.

En esta forma de comunicación, se crea una cola en la cual se depositan los mensajes para luego ser entregados a su destinatario, al igual que el segmento de memoria compartida maneja punteros genéricos para especificar el contenido del mensaje, lo cual resulta de gran provecho ya que no existe restricción con los tipos de datos que se manejan.

3.1. Elección de la llave para la cola de mensajes.

Similar a la memoria compartida, la llave para la creación de la cola de mensajes se puede lograr a través de la función `ftok()` o definiendo una dirección lógica hexadecimal cualquiera. También al igual que en el segmento de memoria compartida se debe definir la estructura que tendrán los mensajes.

3.2. Creación o búsqueda del segmento de memoria compartida.

Luego de haber asignado la dirección que tendrá la cola de mensajes, se puede crear la cola con la siguiente función:

```
int msgget(key_t key, int shmflg);
```

Para usar esta función con el fin de crear un segmento de memoria, se debe pasar como parámetro `shmflg` la permisologia en formato octal y la bandera IPC_CREAT la cual indica que se debe crear la cola. En caso de solo querer obtener el identificador de una determinada cola existente, solo es necesario

enviar como parámetro `shmflg` la permisologia en formato octal. Existen una variedad de configuraciones para el parámetro `shmflg`, sin embargo estas dos serán suficientes por los momentos.

3.3. Recibir un mensaje a través de la cola.

Luego de ser creada la cola de mensajes un proceso interesado en recibir un mensaje puede notificarlo por medio de la siguiente función:

```
size_t msgrcv(int msqid,void *msgp, size_t msgsz,
              long msgtyp, int msgflg);
```

3.4. Enviar un mensaje a través de la cola.

Un proceso puede enviar un mensaje a otro proceso haciendo uso de la siguiente función:

```
int msgsnd(int msqid, const void *msgp, size_t msgsz,
           int msgflg);
```

3.5. Eliminar una cola de mensajes.

Una vez se termine de utilizar la cola de mensajes por cada uno de los procesos involucrados, ésta debe ser eliminada, ya que la culminación de estos procesos no involucra la eliminación automática de la cola de mensajes y por lo tanto los recursos utilizados no se liberaran. Para eliminar la cola de mensajes y liberar la porción de memoria que esta ocupa, se hace uso de la siguiente función:

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

3.6. Implementación de una cola de mensajes.

En el siguiente ejemplo se observa la creación de tres procesos, uno de ellos calcula los términos de la sucesión de Fibonacci, otro muestra los términos pares y el ultimo muestra los términos impares. A continuación se muestra el código fuente:

```
1
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/msg.h>
7 #include <signal.h>
8 #include <errno.h>
```

```c
#include "colamsg.h"

void exit_signal(int);

int main()
{
  system("clear");
  printf("\n_________________________  Sucesion de
     Fibonacci _________________________\n\n");

  key_t key_colamsg = ftok(ROUTE, ID);
  int id_colamsg;
  msg_data data;



  signal(2, exit_signal);

  if((id_colamsg = msgget(key_colamsg, 0)) != -1)
    msgctl(id_colamsg, IPC_RMID, 0);

  if((id_colamsg = msgget(key_colamsg, IPC_CREAT |
     0666)) == -1)
  {
        perror("msgget");
                exit(EXIT_FAILURE);
        }


  int anterior0 = 0;
        int anterior1 = 1;
        int actual = 0;
        int i=3;

        printf("Termino 1: %d\n",0);
        fflush(stdout);
  data.id = 1;
  data.numero = 0;
  if (msgsnd(id_colamsg, (void *)&data, sizeof(
     msg_data), 0) == -1)
  {
          perror("Envio fallido");
  }
        sleep(1);
        printf("Termino 2: %d\n",1);
        fflush(stdout);
```

```
52          data.id = 2;
53    data.numero = 1;
54    if (msgsnd(id_colamsg, (struct msgbuf *)&data,
          sizeof(msg_data), 0) == -1)
55     perror("Envio fallido");
56          sleep(1);
57
58          while(1)
59          {
60          actual = anterior0 + anterior1;
61                  printf("Termino %d: %d\n",i , actual);
62                  fflush(stdout);
63                  anterior0=anterior1;
64                  anterior1=actual;
65                  i++;
66          if(actual%2  == 0)
67                    data.id = 1;
68                  else
69                    data.id = 2;
70      data.numero = actual;
71      if (msgsnd(id_colamsg, (struct msgbuf *)&data,
            sizeof(msg_data), 0) == -1)
72        perror("Envio fallido");
73      sleep(1);
74          }
75
76          return 0;
77  }
78
79  void exit_signal(int num_signal)
80  {
81    key_t key_colamsg = ftok(ROUTE, ID);
82    int id_colamsg;
83    msg_data data;
84
85          if((id_colamsg = msgget(key_colamsg, 0)) ==
                -1)
86          {
87            perror("msgget");
88            exit(EXIT_FAILURE);
89          }
90    if (msgrcv(id_colamsg, (struct msgbuf *)&data,
          sizeof(msg_data), (long)3, 0) == -1)
91    {
92      perror("msgrcv");
93      exit(EXIT_FAILURE);
```

```
  }
  kill(data.numero, SIGINT);
  printf("id: %d, numero:%d\n",data.id, data.numero);

  sleep(1);

        if (msgrcv(id_colamsg, (struct msgbuf *)&data,
            sizeof(msg_data), (long)4, 0) == -1)
  {
    perror("msgrcv");
    exit(EXIT_FAILURE);
  }
  kill(data.numero, SIGINT);

  sleep(2);



        printf("\n\n\tHasta luego!\n");
        fflush(stdout);
  exit(EXIT_SUCCESS);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <errno.h>
#include <signal.h>
#include "colamsg.h"

void exit_signal(int);

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

  if(argc != 2 )
  {
    printf("\n\tCantidad incorrecta de parametros!\n")
        ;
    exit(EXIT_FAILURE);
  }

  int tipo = atoi(argv[1]);

```

```
23    if(tipo != 1 && tipo != 2 )
24    {
25      printf("\n\tParametros incorrectos!\n");
26      exit(EXIT_FAILURE);
27    }
28
29    signal(2, exit_signal);
30
31    key_t key_colamsg = ftok(ROUTE, ID);
32    int id_colamsg;
33    msg_data data;
34
35          if((id_colamsg = msgget(key_colamsg, 0)) ==
                -1)
36          {
37            perror("msgget");
38            exit(EXIT_FAILURE);
39          }
40
41    data.id = (tipo+2);
42    data.numero = getpid();
43    if(msgsnd(id_colamsg, (struct msgbuf *)&data, sizeof
         (msg_data), 0) == -1)
44      perror("Envio fallido");
45
46    printf("id: %d, numero:%d",data.id, data.numero);
47
48
49    if(tipo == 1)
50      printf("\t --> %d Terminos Pares: \n\n\t", getpid
           ());
51    else
52      printf("\t --> %d Terminos Impares: \n\n\t",
           getpid());
53    fflush(stdout);
54
55    while(1)
56    {
57      if(tipo == 1)
58      {
59
60       if(msgrcv(id_colamsg, (struct msgbuf *)&data,
            sizeof(msg_data), (long)1, 0) == -1)
61       {
62         perror("msgrcv");
63         exit(EXIT_FAILURE);
```

```
64        }
65        printf(" %d - ", data.numero);
66        fflush(stdout);
67       }
68       else
69       {
70        if(msgrcv(id_colamsg, (struct msgbuf *)&data,
             sizeof(msg_data), (long)2, 0) == -1)
71        {
72          perror("msgrcv");
73          exit(EXIT_FAILURE);
74        }
75
76        printf(" %d - ", data.numero);
77        fflush(stdout);
78       }
79
80    }
81  }
82
83  void exit_signal(int num_signal)
84  {
85    printf("\n\n\tHasta luego!\n");
86    fflush(stdout);
87    exit(EXIT_SUCCESS);
88  }
```

```
1  #define ID 123
2  #define ROUTE "/bin/lspci"
3
4  typedef struct
5  {
6    long id;
7    long long numero;
8  }msg_data;
```