Trabajo práctico 3: comunicación entre Procesos

Sistemas Operativos

Grupo 18:

Chort, Julio Alberto julioch_17@outlook.com

Pacheco Pilan, Federico Ignacio fedepacheco2112@gmail.com

Reynoso, Valentín valenreynoso17@gmail.com

Segundo Cuatrimestre

2020

1. Resoluciones

1.1. Ejercicio 1: semáforos

inicializarSemaforos.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <linux/ipc.h> // si se coloca "sys" en lugar de "linux" no compila
#include <linux/sem.h> // idem

int main()
{
   key_t claveKit, claveHisopado, claveResultado;
   int semaforoKit, semaforoHisopado, semaforoResultado;
   union semun arg;

   claveKit = ftok(".", 'K');
   claveHisopado = ftok(".", 'H');
   claveResultado = ftok(".", 'R');

         // crear semaforos
   semaforoKit = semget(claveKit, 1, 0666 | IPC_CREAT);
   semaforoHisopado = semget(claveHisopado, 1, 0666 | IPC_CREAT);
   semaforoResultado = semget(claveResultado, 1, 0666 | IPC_CREAT);

         // inicializar semaforos
   arg.val = 0;
   semctl(semaforoKit, 0, SETVAL, arg);

   arg.val = 0;
   semctl(semaforoHisopado, 0, SETVAL, arg);

   arg.val = 1;
   semctl(semaforoResultado, 0, SETVAL, arg);

   return 0;
}
```

kit.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
```

```
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>
#define pacientesPorDia 5
#define cantidadDias 2

int main()
{
   key_t claveKit, claveResultado;
        int semaforoKit, semaforoResultado;
   int i, j;
        struct sembuf sembufDown = {0, -1, 0};
   struct sembuf sembufUp = {0, 1, 0};

        claveKit = ftok(".", 'K');
   claveResultado = ftok(".", 'R');

        semaforoKit = semget(claveKit, 1, 0); // conectarse a los semaforos
creados en inicializarSemaforos.c
        semaforoResultado = semget(claveResultado, 1, 0);

   // Acciones de kit propiamente

   for (i = 1; i <= cantidadDias; i++)
   {
        printf("Dia: %i\n", i);
        for (j = 1; j <= pacientesPorDia; j++)
        {
                semop(semaforoResultado, &sembufDown, 1);   // down(resultado)
                printf("1) Abriendo kit. Preparando... Persona: %i\n", j);
                sleep(1);
                semop(semaforoKit, &sembufUp, 1);               // up(kit)

        }
   }

   return 0;
}
```

hisopado.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>
#define pacientesPorDia 5
```

```
#define cantidadDias 2

int main()
{
   key_t claveHisopado, claveKit;
         int semaforoHisopado, semaforoKit;
   int i, j;
         struct sembuf sembufDown = {0, -1, 0};
   struct sembuf sembufUp = {0, 1, 0};

   claveHisopado = ftok(".", 'H');
   claveKit = ftok(".", 'K');

         semaforoHisopado = semget(claveHisopado, 1, 0); // conectarse a los
semaforos creados en inicializarSemaforos.c
   semaforoKit = semget(claveKit, 1, 0);

   // Acciones de hisopado propiamente
         for (i = 1; i <= cantidadDias; i++)
   {
         printf("Dia: %i\n", i);
         for (j = 1; j <= pacientesPorDia; j++)
         {
                  semop(semaforoKit, &sembufDown, 1);   // down(kit)
                  printf("2) Tomando muestra. Hisopando ... Persona: %i\n", j);
                  sleep(2);
                  semop(semaforoHisopado, &sembufUp, 1);   // up(hisopado)
         }
   }

   return 0;
}
```

resultado.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>
#define pacientesPorDia 5
#define cantidadDias 2

int main()
{
   key_t claveHisopado, claveResultado;
         int semaforoHisopado, semaforoResultado;
```

```
   int i, j;
         struct sembuf sembufDown = {0, -1, 0};
   struct sembuf sembufUp = {0, 1, 0};

   claveHisopado = ftok(".", 'H');
   claveResultado = ftok(".", 'R');

         semaforoHisopado = semget(claveHisopado, 1, 0);  // conectarse a los
semaforos creados en inicializarSemaforos.c
   semaforoResultado = semget(claveResultado, 1, 0);

   // Acciones de resultado propiamente
         for (i = 1; i <= cantidadDias; i++)
   {
         printf("Dia: %i\n", i);
         for (j = 1; j <= pacientesPorDia; j++)
         {
                  semop(semaforoHisopado, &sembufDown, 1);   // down(hisopado)
                  printf("3) Almacenando muestra. Sellando resultado... Persona:
%i\n", j);
                  sleep(3);
                  semop(semaforoResultado, &sembufUp, 1);    // up(resultado)
         }
   }

   return 0;
}
```

eliminarSemaforos.c

```
#include <sys/types.h>
#include <linux/ipc.h>
#include <linux/sem.h>

int main()
{
   semctl( semget( ftok(".", 'K'), 1, 0666 ) , 0, IPC_RMID, 0);
   semctl( semget( ftok(".", 'H'), 1, 0666 ) , 0, IPC_RMID, 0);
   semctl( semget( ftok(".", 'R'), 1, 0666 ) , 0, IPC_RMID, 0);

   return 0;
}
```

1.2. Ejercicio 2: productor - consumidor con mensajes

inicializarColasMensajes.c

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int main()
{
   key_t claveSiembra, claveCosecha;
   int idSiembra, idCosecha;

   claveSiembra = ftok(".", 'S');
   claveCosecha = ftok(".", 'C');

   // crear colas de mensajes
   idSiembra = msgget(claveSiembra, 0666 | IPC_CREAT);
   idCosecha = msgget(claveCosecha, 0666 | IPC_CREAT);

   return 0;
}
```

siembra.c

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define periodosSiembra 15
#define cantidadProductosSinSoja 3

char productosSinSoja[cantidadProductosSinSoja] = {'M', 'L', 'G'};
int contadorSoja = 0;

struct producto_msgbuf
{
   long mtype;
   char nombreProducto;
};

char decidirSiembra();

int main()
{
```

```
   key_t claveSiembra, claveCosecha;
   int idSiembra, idCosecha;
   int i;
   char productoElegido;
   struct producto_msgbuf productoSiembra;

   claveSiembra = ftok(".", 'S');
   claveCosecha = ftok(".", 'C');

   // conectarse
   idSiembra = msgget(claveSiembra, 0666);
   idCosecha = msgget(claveCosecha, 0666);

   srand(time(NULL));

   for (i = 0; i < periodosSiembra; i++)
   {
         msgrcv(idSiembra, &productoSiembra, sizeof(productoSiembra), 0, 0);   //
recibir un mensaje vacio / msgtyp = 0 para recibir de cualquier tipo

         productoElegido = decidirSiembra();
         printf("Se recibio: libre. \nSe enviara a sembrar: %c\n", productoElegido);

         productoSiembra.nombreProducto = productoElegido;

         msgsnd(idCosecha, &productoSiembra, sizeof(productoSiembra), 0);
   }

   return 0;
}

char decidirSiembra()
{
   char ch;

   if (contadorSoja < 3)
   {
         contadorSoja++;
         ch = 'S';
   }
   else
   {
         contadorSoja = 0;
         ch = productosSinSoja[rand() % cantidadProductosSinSoja];
   }

   return ch;
}
```

cosecha.c

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#define parcelasDisponibles 3
#define periodosSiembra 15
#define cantidadProductos 4

struct producto_msgbuf
{
   long mtype;
   char nombreProducto;
};

void cosechar(struct producto_msgbuf*);

int main()
{
   key_t claveSiembra, claveCosecha;
   int idSiembra, idCosecha;
   int i;
   struct producto_msgbuf productoVacio = {1, '\0'};
   struct producto_msgbuf productoCosecha;

   claveSiembra = ftok(".", 'S');
   claveCosecha = ftok(".", 'C');

   // conectarse
   idSiembra = msgget(claveSiembra, 0666);
   idCosecha = msgget(claveCosecha, 0666);

   for (i = 0; i < parcelasDisponibles; i++)
   {
         printf("Aviso: parcela lista para sembrar...\n");
         msgsnd(idSiembra, &productoVacio, sizeof(productoVacio), 0);
   }

   for (i = 0; i < periodosSiembra; i++)
   {
         msgrcv(idCosecha, &productoCosecha, sizeof(productoCosecha), 0, 0);   //
msgtyp = 0 para recibir de cualquier tipo

         cosechar(&productoCosecha);

         msgsnd(idSiembra, &productoCosecha, sizeof(productoCosecha), 0);
   }
```

```
    return 0;
}

void cosechar(struct producto_msgbuf* producto)
{
    int i;

    if (producto -> nombreProducto == 'S')
            printf("Nueva cosecha prioritaria (");
    else
            printf("Se ha cosechado (");

    for (i = 0; i < 10; i++)
            printf("%c", producto -> nombreProducto);
    printf(")\nAviso, porcion de parcela libre para nuevo elemento a sembrar...\n");
}
```

eliminarColasMensajes.c

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int main()
{
    key_t claveSiembra, claveCosecha;
    int idSiembra, idCosecha;

    claveSiembra = ftok(".", 'S');
    claveCosecha = ftok(".", 'C');

    idSiembra = msgget(claveSiembra, 0666);
    idCosecha = msgget(claveCosecha, 0666);

    msgctl(idCosecha, IPC_RMID, 0);
    msgctl(idSiembra, IPC_RMID, 0);

    return 0;
}
```

1.3. Ejercicio 3: memoria compartida y semáforos

inicializar.c

```
#include <stdio.h>
#include <sys/types.h>
#include <linux/ipc.h>
#include <linux/sem.h>
//#include <sys/shm.h>
#define tamanioMemoria 32

int main()
{
   key_t claveClubPrimario, claveClubSecundario, claveBandera;
   int semaforoClubPrimario, semaforoClubSecundario;
   union semun arg;

   claveClubPrimario = ftok(".", 'P');
   claveClubSecundario = ftok(".", 'S');
   claveBandera = ftok(".", 'B');

   // crear e inicializar semaforos clubes
   semaforoClubPrimario = semget(claveClubPrimario, 1, 0666 | IPC_CREAT);
   semaforoClubSecundario = semget(claveClubSecundario, 1, 0666 | IPC_CREAT);


   arg.val = 1;
   semctl(semaforoClubPrimario, 0, SETVAL, arg);
   arg.val = 0;
   semctl(semaforoClubSecundario, 0, SETVAL, arg);

   // crear memoria compartida
   shmget(claveBandera, tamanioMemoria, IPC_CREAT | 0666);

                return 0;
}
```

clubPrimario.c

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

```
#define cantidadColores 3
#define tamanioCadenaColor 4
#define tamanioMemoria 32

char colores [cantidadColores][tamanioCadenaColor] = {"Roj", "Ama", "Azu"};

int main ()
{
   key_t claveClubPrimario, claveClubSecundario, claveBandera,
idMemoriaCompartida;
   int semaforoClubPrimario, semaforoClubSecundario;
   int i;
   char* memoriaCompartidaPtr;
   struct sembuf sembufDown = {0, -1, 0};
   struct sembuf sembufUp = {0, 1, 0};

   claveClubPrimario = ftok(".", 'P');
   claveClubSecundario = ftok(".", 'S');
   claveBandera = ftok(".", 'B');

   semaforoClubPrimario = semget(claveClubPrimario, 1, 0);   // conectarse al
semaforo
   semaforoClubSecundario = semget(claveClubSecundario, 1, 0);

   // conectar el proceso al segmento
   idMemoriaCompartida = shmget(claveBandera, tamanioMemoria, IPC_CREAT);
   memoriaCompartidaPtr = shmat(idMemoriaCompartida, 0, 0);

   for (i = 0; i < cantidadColores; i++)
   {
         // entrar a la "region critica"
         semop(semaforoClubPrimario, &sembufDown, 1);
         strcat(memoriaCompartidaPtr, colores[i]);
         sleep(1);       // retraso para que se pueda visualizar como se van
agregando los colores
         semop(semaforoClubSecundario, &sembufUp, 1);
         // salir

         printf("%s\n", colores[i]);
   }

   return 0;
}
```

clubSecundario.c

```
#include <stdio.h>
#include <string.h>
```

```
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#define cantidadColores 3
#define tamanioCadenaColor 4
#define tamanioMemoria 32

char colores [cantidadColores][tamanioCadenaColor] = {"Nar", "Ver", "Vio"};

int main ()   // similar a clubPrimario.c, salvo por los semaforos y colores
{
   key_t claveClubPrimario, claveClubSecundario, claveBandera,
idMemoriaCompartida;
   int semaforoClubPrimario, semaforoClubSecundario;
   int i;
   char* memoriaCompartidaPtr;
   struct sembuf sembufDown = {0, -1, 0};
   struct sembuf sembufUp = {0, 1, 0};

   claveClubPrimario = ftok(".", 'P');
   claveClubSecundario = ftok(".", 'S');
   claveBandera = ftok(".", 'B');

   semaforoClubPrimario = semget(claveClubPrimario, 1, 0);   // conectarse al
semaforo
   semaforoClubSecundario = semget(claveClubSecundario, 1, 0);

   // conectar el proceso al segmento
   idMemoriaCompartida = shmget(claveBandera, tamanioMemoria, IPC_CREAT);
   memoriaCompartidaPtr = shmat(idMemoriaCompartida, 0, 0);

   for (i = 0; i < cantidadColores; i++)
   {
         // entrar a la "region critica"
         semop(semaforoClubSecundario, &sembufDown, 1);
         strcat(memoriaCompartidaPtr, colores[i]);
         sleep(1);       // retraso para que se pueda visualizar como se van
agregando los colores
         semop(semaforoClubPrimario, &sembufUp, 1);
         // salir

         printf("%s\n", colores[i]);
   }

   printf("\nBandera final: %s\n", memoriaCompartidaPtr);
```

```
   return 0;
}
```

finalizar.c

```
#include <stdio.h>
#include <sys/types.h>
#include <linux/ipc.h>
#include <linux/sem.h>
//#include <sys/shm.h>
#define tamanioMemoria 32

int main()
{
   semctl( semget( ftok(".", 'P'), 1, 0666) , 0, IPC_RMID, 0);
   semctl( semget( ftok(".", 'S'), 1, 0666) , 0, IPC_RMID, 0);
   shmctl( shmget( ftok(".", 'B'), tamanioMemoria, 0666 ) , IPC_RMID, 0);

                        return 0;
}
```

2. Consideraciones

2.1. De la resolución:

Ejercicio 1:

- Por temas de simplificación del problema, se han hecho dos archivos llamados ‘inicializarSemaforos.c’ y ‘eliminarSemaforos.c’ que deben ser ejecutados antes y después de los predicados propuestos en el enunciado, respectivamente. Por lo tanto, el orden de ejecución deberá ser el siguiente:
  - ‘inicializarSemaforos.c’
  - ‘kit.c’, ‘hisopado.c’ y ‘resultado.c’ (en cualquier orden)
  - ‘eliminarSemaforos.c’
- En una ocasión probando el ejercicio nos topamos con un error que no pudimos determinar ni volver a repetir. Éste consistía en que al ejecutar, por ejemplo, ‘kit.c’ se “abrían” todos los kits de las cinco personas de los dos días, sin bloquearse el proceso (no esperaba que ejecutásemos ‘hisopado.c’ y ‘resultado.c’). No sabemos porqué se produjo ni si podría volver a pasar.

Ejercicio 2:

- Por temas de simplificación del problema, se han hecho dos archivos llamados ‘inicializarColasMensajes.c’ y ‘eliminarColasMensajes.c’ que deben ser ejecutados antes y después de los predicados propuestos en el enunciado, respectivamente. Por lo tanto, el orden de ejecución deberá ser el siguiente:
  - ‘inicializarColasMensajes.c’
  - ‘siembra.c’ y ‘cosecha.c’ (en cualquier orden)
  - ‘eliminarColasMensajes.c’

Ejercicio 3:

- Por temas de simplificación del problema, se han hecho dos archivos llamados ‘inicializar.c’ y ‘finalizar.c’ que deben ser ejecutados antes y después de los predicados propuestos en el enunciado, respectivamente. Por lo tanto, el orden de ejecución deberá ser el siguiente:
  - ‘inicializar.c’

- ‘clubPrimario.c’ y ‘clubSecundario.c’ (en cualquier orden)
- ‘finalizar.c’

- Se ha agregado un comando **sleep(1)** en los códigos de los ejercicios ‘clubPrimario.c’ y ‘clubSecundario.c’ para que al ejecutarse pueda verse bien la salida y las lecturas/escrituras en la memoria compartida.

2.2. Fuentes consultadas:[1]

Ejercicio 1: -

Ejercicio 2: -

Ejercicio 3: -

Misceláneo: -

[1] En general, no se citan las entradas de manual consultadas. No se cita el material proporcionado por la cátedra.