secuencial_pipe.c:

```
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include "constantes.h"

int pa_b[2], pb_c[2], pc_d[2], pd_e[2], pe_a[2];

void leer(int pipe, char * caracter, int size) {
	int leidos = read(pipe,caracter,size);
	if ( leidos == -1 ) {
		printf("Error al leer del pipe.\n");
		exit(error_leer_pipe);
	}
}

void escribir(int pipe, char * caracter, int size) {
	int escritos = write(pipe,caracter,size);
	if ( escritos == -1 ) {
		printf("Error al escribir en el pipe.\n");
		exit(error_escribir_pipe);
	}
}

void A() {
	if ( close(pa_b[0]) == -1 && close(pe_a[1]) == -1 ) {
		printf("Error al cerrar el pipe.\n");
		exit(error_close_pipe);
	}

	char p;
	while (1) {
		leer(pe_a[0],&p,1);
		printf("A\n");
		escribir(pa_b[1],"B",1);
	}
}

void B() {
	if ( close(pb_c[0]) == -1 && close(pa_b[1]) == -1 ) {
		printf("Error al cerrar el pipe.\n");
		exit(error_close_pipe);
	}

	char p;
	while (1) {
		leer(pa_b[0],&p,1);
		printf("B\n");
		escribir(pb_c[1],"C",1);
	}
}

void C() {
	if ( close(pc_d[0]) == -1 && close(pb_c[1]) == -1 ) {
		printf("Error al cerrar el pipe.\n");
		exit(error_close_pipe);
	}

	char p;
	while (1) {
		leer(pb_c[0],&p,1);
		printf("C\n");
		escribir(pc_d[1],"D",1);
	}
}

void D() {
	if ( close(pd_e[0]) == -1 && close(pc_d[1]) == -1 ) {
		printf("Error al cerrar el pipe.\n");
```

```
            exit(error_close_pipe);
        }

        char p;
        while (1) {
            leer(pc_d[0],&p,1);
            printf("D\n");
            escribir(pd_e[1],"E",1);
        }
}

void E() {
    if ( close(pe_a[0]) == -1 && close(pd_e[1]) == -1 ) {
            printf("Error al cerrar el pipe.\n");
            exit(error_close_pipe);
        }
        char p;
        while (1) {
            leer(pd_e[0],&p,1);
            printf("E\n");
            escribir(pe_a[1],"A",1);
        }
}

int main() {
      if ( !pipe(pa_b) && !pipe(pb_c) && !pipe(pc_d) && !pipe(pd_e) && !pipe(pe_a) ) {
            int pid;

          escribir(pe_a[1],"A",1);
            pid = fork();
            if ( !pid ) { //hijo
                A();
                exit(0);
            }
            else if (pid < 0) {
                printf("Error en la creación de procesos.\n");
                exit(error_fork);
            }

            pid = fork();
            if ( !pid) { //hijo
                B();
                exit(0);
            }
            else if (pid < 0) {
                printf("Error en la creación de procesos.\n");
                exit(error_fork);
            }

pid = fork();
if ( !pid) { //hijo
      C();
      exit(0);
}
else if (pid < 0) {
      printf("Error en la creación de procesos.\n");
      exit(error_fork);
}

pid = fork();
if ( !pid) { //hijo
      D();
      exit(0);
}
else if (pid < 0) {
      printf("Error en la creación de procesos.\n");
      exit(error_fork);
}

pid = fork();
if ( !pid) { //hijo
      E();
```

```
                exit(0);
            }
            else if (pid < 0) {
                printf("Error en la creación de procesos.\n");
                exit(error_fork);
            }

            for ( int i = 0; i < 5; i++) wait(NULL);
        }
        else {
            printf("Error al crear el pipe.\n");
            exit(error_pipe);
        }
        return 0;
}
```

secuencia2_pipe.c:

```
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include "constantes.h"

int pabc_abc[2], pabc_d[2], pd_e[2], pe_abc[2];

void escribir(int pipe, char * caracter, int size) {
      int escritos = write(pipe,caracter,size);
      if ( escritos == -1 ) {
            printf("Error al escribir en el pipe.\n");
            exit(error_escribir_pipe);
      }
}

void A() {
      if ( close(pabc_d[0]) == -1 && close(pe_abc[1]) == -1 ) {
            printf("Error al cerrar el pipe.\n");
            exit(error_close_pipe);
      }

      char p;
    int leidos;
      while (1) {
            leidos = read(pe_abc[0],&p,1);
       if ( leidos != -1 ) {
           printf("A\n");
           escribir(pabc_abc[1],"X",1);
       }
       leidos = read(pabc_abc[0],&p,1);
       if ( leidos != -1 ) {
           printf("A\n");
           escribir(pabc_d[1],"D",1);
       }
      }
}

void B() {
      if ( close(pabc_d[0]) == -1 && close(pe_abc[1]) == -1 ) {
            printf("Error al cerrar el pipe.\n");
            exit(error_close_pipe);
      }

      char p;
    int leidos;
      while (1) {
            leidos = read(pe_abc[0],&p,1);
       if ( leidos != -1 ) {
           printf("B\n");
           escribir(pabc_abc[1],"X",1);
       }
       leidos = read(pabc_abc[0],&p,1);
```

```
        if ( leidos != -1 ) {
            printf("B\n");
            escribir(pabc_d[1],"D",1);
        }
       }
}

void C() {
      if ( close(pabc_d[0]) == -1 && close(pe_abc[1]) == -1 ) {
            printf("Error al cerrar el pipe.\n");
            exit(error_close_pipe);
      }

      char p;
    int leidos;
      while (1) {
            leidos = read(pe_abc[0],&p,1);
        if ( leidos != -1 ) {
            printf("C\n");
            escribir(pabc_abc[1],"X",1);
        }
        leidos = read(pabc_abc[0],&p,1);
        if ( leidos != -1 ) {
            printf("C\n");
            escribir(pabc_d[1],"D",1);
         }
       }
}

void D() {
      if ( close(pd_e[0]) == -1 && close(pabc_d[1]) == -1 ) {
            printf("Error al cerrar el pipe.\n");
            exit(error_close_pipe);
      }

      char p;
      while (1) {
            read(pabc_d[0],&p,1);
            printf("D\n");
            escribir(pd_e[1],"E",1);
      }
}

void E() {
      if ( close(pe_abc[0]) == -1 && close(pd_e[1]) == -1 ) {
            printf("Error al cerrar el pipe.\n");
            exit(error_close_pipe);
      }

      char p;
      while (1) {
            read(pd_e[0],&p,1);
            printf("E\n");
            escribir(pe_abc[1],"X",1);
      }
}

int main() {
      if ( !pipe(pabc_abc) && !pipe(pabc_d) && !pipe(pd_e) && !pipe(pe_abc) ) {
            int pid;

        if ( !fcntl(pe_abc[0], F_SETFL, O_NONBLOCK) && !fcntl(pabc_abc[0], F_SETFL, O_NONBLOCK) ) {

            escribir(pe_abc[1],"X",1);
            pid = fork();
            if ( !pid ) { //hijo
                A();
                exit(0);
            }
            else if (pid < 0) {
                printf("Error en la creación de procesos.\n");
                exit(error_fork);
```

```
}

pid = fork();
if ( !pid ) { //hijo
    B();
    exit(0);
}
else if (pid < 0) {
    printf("Error en la creación de procesos.\n");
    exit(error_fork);
}

pid = fork();
if ( !pid ) { //hijo
    C();
    exit(0);
}
else if (pid < 0) {
    printf("Error en la creación de procesos.\n");
    exit(error_fork);
}

pid = fork();
if ( !pid ) { //hijo
    D();
    exit(0);
}
else if (pid < 0) {
    printf("Error en la creación de procesos.\n");
    exit(error_fork);
}

pid = fork();
if ( !pid ) { //hijo
    E();
    exit(0);
}
else if (pid < 0) {
    printf("Error en la creación de procesos.\n");
    exit(error_fork);
}

for ( int i = 0; i < 5; i++) wait(NULL);
 }
 else {
    printf("Error al crear los pipes.\n");
    exit(error_pipe);
 }
}
    else {
        printf("Error al crear los pipes.\n");
        exit(error_pipe);
    }
    return 0;
}
```

secuencia3_pipe.c:

```
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include "constantes.h"

int pa_c[2], pc_d[2], pd_e[2], pe_b[2], pb_c[2], pe_a[2], pa_b[2];

void leer(int pipe, char * caracter, int size) {
    int leidos = read(pipe,caracter,size);
    if ( leidos == -1 ) {
        printf("Error al leer del pipe.\n");
        exit(error_leer_pipe);
```

```
	}
}

void escribir(int pipe, char * caracter, int size) {
	int escritos = write(pipe,caracter,size);
	if ( escritos == -1 ) {
		printf("Error al escribir en el pipe.\n");
		exit(error_escribir_pipe);
	}
}

void A() {
	if ( close(pa_b[0]) == -1 && close(pe_a[1]) == -1 ) {
		printf("Error al cerrar el pipe.\n");
		exit(error_close_pipe);
	}

	char p;
	while (1) {
		leer(pe_a[0],&p,1);
		printf("A\n");
		escribir(pa_c[1],"C",1);
		leer(pe_a[0],&p,1);
		printf("A\n");
		escribir(pa_b[1],"B",1);
	}
}

void B() {
	if ( close(pb_c[0]) == -1 && close(pa_b[1]) == -1 ) {
		printf("Error al cerrar el pipe.\n");
		exit(error_close_pipe);
	}

	char p;
	while (1) {
		leer(pe_b[0],&p,1);
		printf("B\n");
		escribir(pb_c[1],"C",1);
		leer(pa_b[0],&p,1);
		printf("B\n");
		escribir(pb_c[1],"C",1);
	}
}

void C() {
	if ( close(pc_d[0]) == -1 && close(pb_c[1]) == -1 ) {
		printf("Error al cerrar el pipe.\n");
		exit(error_close_pipe);
	}

	char p;
    int i;
	while (1) {
		leer(pa_c[0],&p,1);
		printf("C\n");
		escribir(pc_d[1],"D",1);
		for ( i = 0; i < 2; i++) {
		leer(pb_c[0],&p,1);
		printf("C\n");
		escribir(pc_d[1],"D",1);
	 }
	}
}

void D() {
	if ( close(pd_e[0]) == -1 && close(pc_d[1]) == -1 ) {
		printf("Error al cerrar el pipe.\n");
		exit(error_close_pipe);
	}

	char p;
```

```
        while (1) {
                leer(pc_d[0],&p,1);
                printf("D\n");
                escribir(pd_e[1],"E",1);
        }
}

void E() {
        if ( close(pe_a[0]) == -1 && close(pd_e[1]) == -1 ) {
                printf("Error al cerrar el pipe.\n");
                exit(error_close_pipe);
        }

        char p;
      int i;
        while (1) {
                leer(pd_e[0],&p,1);
                printf("E\n");
         escribir(pe_b[1],"B",1);
         for(i=0; i<2; i++){
             leer(pd_e[0],&p,1);
             printf("E\n");
             escribir(pe_a[1],"A",1);
          }
        }
}

int main() {
        if ( !pipe(pa_c) && !pipe(pc_d) && !pipe(pd_e) && !pipe(pe_b) && !pipe(pb_c) && !pipe(pe_a) && !pipe(pa_b) ) {
                int pid;

             escribir(pe_a[1],"A",1);
                pid = fork();
                if ( !pid ) { //hijo
                        A();
                        exit(0);
                }
                else if (pid < 0) {
                        printf("Error en la creación de procesos.\n");
                        exit(error_fork);
                }

                pid = fork();
                if ( !pid ) { //hijo
                        B();
                        exit(0);
                }
                else if (pid < 0) {
                        printf("Error en la creación de procesos.\n");
                        exit(error_fork);
}

pid = fork();
if ( !pid ) { //hijo
        C();
        exit(0);
}
else if (pid < 0) {
        printf("Error en la creación de procesos.\n");
        exit(error_fork);
}

pid = fork();
if ( !pid ) { //hijo
        D();
        exit(0);
}
else if (pid < 0) {
        printf("Error en la creación de procesos.\n");
        exit(error_fork);
}
```

```
            pid = fork();
            if ( !pid ) { //hijo
                E();
                exit(0);
            }
            else if (pid < 0) {
                printf("Error en la creación de procesos.\n");
                exit(error_fork);
            }

            for ( int i = 0; i < 5; i++) wait(NULL);
        }
        else {
            printf("Error al crear el pipe.\n");
            exit(error_pipe);
        }
        return 0;
}
```

secuencia2_colas.c:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "constantes.h"

struct mensaje{
    long tipo;
    int dato;
};

void enviar_mensaje(int id_cola, struct mensaje * m) {
    int error = msgsnd(id_cola,m,sizeof(struct mensaje)-sizeof(long),0);
    if ( error == -1 ) {
            printf("Error al enviar un mensaje a la cola.\n");
            exit(error_send);
    }
}

int recibir_mensaje(int id_cola, struct mensaje * m, long tipo, int flag) {
    int leidos = msgrcv(id_cola,m,sizeof(struct mensaje)-sizeof(long),tipo,flag);
    if ( flag != IPC_NOWAIT && leidos == -1 ) {
            printf("Error al recibir un mensaje de la cola.\n");
            exit(error_receive);
    }
    return leidos;
}

void A(int id_cola) {
    struct mensaje * m = (struct mensaje *) malloc(sizeof(struct mensaje));
    if(m==NULL) {
        printf("Error en malloc.\n");
        exit(error_malloc);
    }
    int leidos = 0;
    while (1) {
        leidos = recibir_mensaje(id_cola,m,EABC,IPC_NOWAIT);
        if ( leidos != -1 ) {
            printf("A\n");
            m->tipo = ABCABC;
            enviar_mensaje(id_cola,m);
        }
        leidos = recibir_mensaje(id_cola,m,ABCABC,IPC_NOWAIT);
        if ( leidos != -1 ) {
            printf("A\n");
            m->tipo = ABCD;
            enviar_mensaje(id_cola,m);
        }
    }
```

```
    free(m);
}

void B(int id_cola) {
    struct mensaje * m = (struct mensaje *) malloc(sizeof(struct mensaje));
    if(m==NULL) {
        printf("Error en malloc.\n");
        exit(error_malloc);
    }
    int leidos = 0;
    while (1) {
        leidos = recibir_mensaje(id_cola,m,EABC,IPC_NOWAIT);
        if ( leidos != -1 ) {
            printf("B\n");
            m->tipo = ABCABC;
            enviar_mensaje(id_cola,m);
        }
        leidos = recibir_mensaje(id_cola,m,ABCABC,IPC_NOWAIT);
        if ( leidos != -1 ) {
            printf("B\n");
            m->tipo = ABCD;
            enviar_mensaje(id_cola,m);
        }
    }
    free(m);
}
void C(int id_cola) {
    struct mensaje * m = (struct mensaje *) malloc(sizeof(struct mensaje));
    if(m==NULL) {
        printf("Error en malloc.\n");
        exit(error_malloc);
    }
    int leidos = 0;
    while (1) {
        leidos = recibir_mensaje(id_cola,m,EABC,IPC_NOWAIT);
        if ( leidos != -1 ) {
            printf("C\n");
            m->tipo = ABCABC;
            enviar_mensaje(id_cola,m);
        }
        leidos = recibir_mensaje(id_cola,m,ABCABC,IPC_NOWAIT);
        if ( leidos != -1 ) {
            printf("C\n");
            m->tipo = ABCD;
            enviar_mensaje(id_cola,m);
        }
    }
    free(m);
}

void D(int id_cola) {
    struct mensaje * m = (struct mensaje *) malloc(sizeof(struct mensaje));
    if(m==NULL) {
        printf("Error en malloc.\n");
        exit(error_malloc);
    }
    while (1) {
        recibir_mensaje(id_cola,m,ABCD,0);
        printf("D\n");
        m->tipo = DE;
        enviar_mensaje(id_cola,m);
    }
    free(m);
}

void E(int id_cola) {
    struct mensaje * m = (struct mensaje *) malloc(sizeof(struct mensaje));
    if(m==NULL) {
        printf("Error en malloc.\n");
        exit(error_malloc);
    }
    while (1) {
```

```
        recibir_mensaje(id_cola,m,DE,0);
        printf("E\n");
        m->tipo = EABC;
        enviar_mensaje(id_cola,m);
    }
    free(m);
}

void inicializar_cola(int id_cola) {
    struct mensaje * m = (struct mensaje *) malloc(sizeof(struct mensaje));
    if(m==NULL) {
        printf("Error en malloc.\n");
        exit(error_malloc);
    }
    m->tipo = EABC;
    m->dato = 0;
    enviar_mensaje(id_cola,m);
    free(m);
}

int main() {
    int id_cola = msgget((key_t) 1234,IPC_CREAT|0666);
    if ( id_cola == -1 ) {
    printf("Error al crear la cola de mensajes.\n");
    exit(error_cola);
}
inicializar_cola(id_cola);
int pid;

pid = fork();
if ( !pid ) { //hijo
    id_cola = msgget((key_t) 1234, 0666);
    if ( id_cola == -1 ) {
        printf("Error al crear la cola de mensajes.\n");
        exit(error_cola);
    }
    A(id_cola);
    exit(0);
}
else if (pid < 0) {
    printf("Error en la creación de procesos.\n");
    exit(error_fork);
}

pid = fork();
if ( !pid ) { //hijo
    id_cola = msgget((key_t) 1234, 0666);
    if ( id_cola == -1 ) {
        printf("Error al crear la cola de mensajes.\n");
        exit(error_cola);
    }
    B(id_cola);
    exit(0);
}
else if (pid < 0) {
    printf("Error en la creación de procesos.\n");
    exit(error_fork);
}

pid = fork();
if ( !pid ) { //hijo
    id_cola = msgget((key_t) 1234, 0666);
    if ( id_cola == -1 ) {
        printf("Error al crear la cola de mensajes.\n");
        exit(error_cola);
    }
    C(id_cola);
    exit(0);
}
else if (pid < 0) {
    printf("Error en la creación de procesos.\n");
    exit(error_fork);
```

```
}

pid = fork();
if ( !pid ) { //hijo
   id_cola = msgget((key_t) 1234, 0666);
   if ( id_cola == -1 ) {
      printf("Error al crear la cola de mensajes.\n");
      exit(error_cola);
   }
   D(id_cola);
   exit(0);
}
else if (pid < 0) {
   printf("Error en la creación de procesos.\n");
   exit(error_fork);
}

pid = fork();
if ( !pid ) { //hijo
   id_cola = msgget((key_t) 1234, 0666);
   if ( id_cola == -1 ) {
      printf("Error al crear la cola de mensajes.\n");
      exit(error_cola);
      }
      E(id_cola);
      exit(0);
   }
   else if (pid < 0) {
      printf("Error en la creación de procesos.\n");
      exit(error_fork);
   }

   for ( int i = 0; i < 5; i++) wait(NULL);
     return 0;
}
```

secuencia3_colas.c:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "constantes.h"

struct mensaje{
   long tipo;
   int dato;
};

void enviar_mensaje(int id_cola, struct mensaje * m) {
   int error = msgsnd(id_cola,m,sizeof(struct mensaje)-sizeof(long),0);
   if ( error == -1 ) {
         printf("Error al enviar un mensaje a la cola.\n");
         exit(error_send);
   }
}

void recibir_mensaje(int id_cola, struct mensaje * m, long tipo) {
   int error = msgrcv(id_cola,m,sizeof(struct mensaje)-sizeof(long),tipo,0);
   if ( error == -1 ) {
         printf("Error al recibir un mensaje de la cola.\n");
         exit(error_receive);
   }
}

void A(int id_cola) {
   struct mensaje * m = (struct mensaje *) malloc(sizeof(struct mensaje));
   if(m==NULL) {
      printf("Error en malloc.\n");
```

```
        exit(error_malloc);
    }
    int envio = 0;
    while (1) {
        recibir_mensaje(id_cola,m,EA);
        printf("A\n");
        if ( !envio ) {
            m->tipo = AC;
            enviar_mensaje(id_cola,m);
            envio = 1;
        }
        else {
            m->tipo = AB;
            enviar_mensaje(id_cola,m);
            envio = 0;
        }
    }
    free(m);
}

void B(int id_cola) {
    struct mensaje * m = (struct mensaje *) malloc(sizeof(struct mensaje));
    if(m==NULL) {
        printf("Error en malloc.\n");
        exit(error_malloc);
    }
    int envio = 0;
    while (1) {
        printf("B\n");
        if ( !envio ) {
            recibir_mensaje(id_cola,m,EB);
            envio = 1;
        }
        else {
            recibir_mensaje(id_cola,m,AB);
            envio = 0;
        }
        m->tipo = BC;
        enviar_mensaje(id_cola,m);
    }
    free(m);
}
void C(int id_cola) {
    struct mensaje * m = (struct mensaje *) malloc(sizeof(struct mensaje));
    if(m==NULL) {
        printf("Error en malloc.\n");
        exit(error_malloc);
    }
    int envio = 0;
    while (1) {
        printf("C\n");
        if ( !envio ) {
            recibir_mensaje(id_cola,m,AC);
            envio = 1;
        }
        else {
            recibir_mensaje(id_cola,m,BC);
            if ( envio == 1 ) envio++;
            else envio = 0;
        }
        m->tipo = CD;
        enviar_mensaje(id_cola,m);
    }
    free(m);
}

void D(int id_cola) {
    struct mensaje * m = (struct mensaje *) malloc(sizeof(struct mensaje));
    if(m==NULL) {
        printf("Error en malloc.\n");
        exit(error_malloc);
    }
```

```
    while (1) {
        recibir_mensaje(id_cola,m,CD);
        printf("D\n");
        m->tipo = DE;
        enviar_mensaje(id_cola,m);
    }
    free(m);
}

void E(int id_cola) {
    struct mensaje * m = (struct mensaje *) malloc(sizeof(struct mensaje));
    if(m==NULL) {
        printf("Error en malloc.\n");
        exit(error_malloc);
    }
    int envio = 0;
    while (1) {
        recibir_mensaje(id_cola,m,DE);
        printf("E\n");
        if ( !envio ) {
            m->tipo = EB;
            enviar_mensaje(id_cola,m);
            envio = 1;
        }
        else {
            m->tipo = EA;
            enviar_mensaje(id_cola,m);
            if ( envio == 1 ) envio++;
            else envio = 0;
        }
    }
    free(m);
}

void inicializar_cola(int id_cola) {
    struct mensaje * m = (struct mensaje *) malloc(sizeof(struct mensaje));
    if(m==NULL) {
        printf("Error en malloc.\n");
        exit(error_malloc);
    }
    m->tipo = EA;
    m->dato = 0;
    enviar_mensaje(id_cola,m);
    free(m);
}

int main() {
    int id_cola = msgget((key_t) 1234,IPC_CREAT|0666);
    if ( id_cola == -1 ) {
    printf("Error al crear la cola de mensajes.\n");
    exit(error_cola);
}
inicializar_cola(id_cola);
int pid;

pid = fork();
if ( !pid ) { //hijo
    id_cola = msgget((key_t) 1234, 0666);
    if ( id_cola == -1 ) {
        printf("Error al crear la cola de mensajes.\n");
        exit(error_cola);
    }
    A(id_cola);
    exit(0);
}
else if (pid < 0) {
    printf("Error en la creación de procesos.\n");
    exit(error_fork);
}

pid = fork();
if ( !pid ) { //hijo
```

```
    id_cola = msgget((key_t) 1234, 0666);
    if ( id_cola == -1 ) {
        printf("Error al crear la cola de mensajes.\n");
        exit(error_cola);
    }
    B(id_cola);
    exit(0);
}
else if (pid < 0) {
    printf("Error en la creación de procesos.\n");
    exit(error_fork);
}

pid = fork();
if ( !pid ) { //hijo
    id_cola = msgget((key_t) 1234, 0666);
    if ( id_cola == -1 ) {
        printf("Error al crear la cola de mensajes.\n");
        exit(error_cola);
    }
    C(id_cola);
    exit(0);
}

else if (pid < 0) {
    printf("Error en la creación de procesos.\n");
    exit(error_fork);
}

pid = fork();
if ( !pid ) { //hijo
    id_cola = msgget((key_t) 1234, 0666);
    if ( id_cola == -1 ) {
        printf("Error al crear la cola de mensajes.\n");
        exit(error_cola);
    }
    D(id_cola);
    exit(0);
}
else if (pid < 0) {
    printf("Error en la creación de procesos.\n");
    exit(error_fork);
}

pid = fork();
if ( !pid ) { //hijo
    id_cola = msgget((key_t) 1234, 0666);
    if ( id_cola == -1 ) {
        printf("Error al crear la cola de mensajes.\n");
        exit(error_cola);
    }
    E(id_cola);
    exit(0);
}
else if (pid < 0) {
    printf("Error en la creación de procesos.\n");
    exit(error_fork);
}

for ( int i = 0; i < 5; i++) wait(NULL);
  return 0;
}
```

oso_abejas_colas.c:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "constantes.h"
```

```
struct mensaje{
    long tipo;
    int porciones_restantes;
};

void enviar_mensaje(int id_cola, struct mensaje * m) {
    int error = msgsnd(id_cola,m,sizeof(struct mensaje)-sizeof(long),0);
    if ( error == -1 ) {
            printf("Error al enviar un mensaje a la cola.\n");
            exit(error_send);
    }
}

void recibir_mensaje(int id_cola, struct mensaje * m, long tipo) {
    int error = msgrcv(id_cola,m,sizeof(struct mensaje)-sizeof(long),tipo,0);
    if ( error == -1 ) {
            printf("Error al recibir un mensaje de la cola.\n");
            exit(error_receive);
    }
}

void oso(int id_cola) {
    struct mensaje * m = (struct mensaje *) malloc(sizeof(struct mensaje));
    if(m==NULL) {
        printf("Error en malloc.\n");
        exit(error_malloc);
    }
    while (1) {
        recibir_mensaje(id_cola,m,2);
        printf("El oso se despierta y empieza a comer\n");
        m->porciones_restantes = capacidad_tarro;
        printf("El tarro está lleno\n");
        m->tipo = 1;
        enviar_mensaje(id_cola,m);
    }
    free(m);
}

void producir_miel(int i) {
    sleep(1);
    printf("La abeja %i produjo una porción de miel.\n",i);
}

void abeja(int i, int id_cola) {
    struct mensaje * m = (struct mensaje *) malloc(sizeof(struct mensaje));
    if(m==NULL) {
        printf("Error en malloc.\n");
        exit(error_malloc);
    }
    while (1) {
        producir_miel(i);
        recibir_mensaje(id_cola,m,1);
        if ( !m->porciones_restantes ) {
            printf("La abeja %i no puede insertar su porción y despierta al oso.\n",i);
            m->tipo = 2;
            enviar_mensaje(id_cola,m);
        }
        else {
            m->porciones_restantes--;
            printf("La abeja %i insertó una porción. Quedan %i.\n",i,m->porciones_restantes);
            m->tipo = 1;
            enviar_mensaje(id_cola,m);
        }
    }
    free(m);
}

void inicializar_cola(int id_cola) {
    struct mensaje * m = (struct mensaje *) malloc(sizeof(struct mensaje));
    if(m==NULL) {
        printf("Error en malloc.\n");
```

```
        exit(error_malloc);
    }
    m->tipo = 1;
    m->porciones_restantes= capacidad_tarro;
    printf("las porciones iniciales son %i\n",m->porciones_restantes);
    enviar_mensaje(id_cola,m);
    free(m);
}


int main() {
    int id_cola = msgget((key_t) 1234,IPC_CREAT|0666);
    if ( id_cola == -1 ) {
        printf("Error al crear la cola de mensajes.\n");
        exit(error_cola);
    }
    inicializar_cola(id_cola);

    int i, pid;
    pid = fork();
    if ( !pid ) {
        id_cola = msgget((key_t) 1234, 0666);
        if ( id_cola == -1 ) {
            printf("Error al crear la cola de mensajes.\n");
            exit(error_cola);
        }
        oso(id_cola);
        exit(0);
    }
    else if ( pid == -1 ) {
        printf("Error en fork.\n");
        exit(error_fork);
    }

    for ( i = 0; i < cantidad_abejas; i++ ) {
        pid = fork();
        if ( !pid ) {
            id_cola = msgget((key_t) 1234, 0666);
            if ( id_cola == -1 ) {
                printf("Error al crear la cola de mensajes.\n");
                exit(error_cola);
            }
            abeja(i,id_cola);
            exit(0);
        }
        else if ( pid == -1 ) {
            printf("Error en fork.\n");
            exit(error_fork);
        }

    for ( i = 0; i < cantidad_abejas+1; i++ ) wait(NULL);
    return 0;
}
```

oso_abejas_shm.c:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <semaphore.h>
#include "constantes.h"

typedef struct memoria_compartida {
    sem_t mutex_tarro, despertar, vacio;
    int porciones_ocupadas;
} * memoria_compartida;

void producir_miel(int i) {
```

```
    sleep(1);
    printf("La abeja %i produjo una porción de miel.\n",i);
}

void oso(memoria_compartida mem) {
    while (1) {
        sem_wait(&mem->despertar);
        printf("El oso se despierta.\n");
        for ( int i = 0; i < capacidad_tarro; i++ ) {
            mem->porciones_ocupadas--;
            printf("El oso come una porción. Quedan %i porciones por comer.\n", mem->porciones_ocupadas);
            sem_post(&mem->vacio);
        }
        sem_post(&mem->mutex_tarro);
    }
}

void abeja(int i, memoria_compartida mem) {
    while (1) {
        producir_miel(i);
        sem_wait(&mem->mutex_tarro);
        if ( !sem_trywait(&mem->vacio) ) {
                mem->porciones_ocupadas++;
                printf("La abeja %i insertó una porción en el tarro. Quedan %i porciones libres.\n",i,capacidad_tarro-mem-
>porciones_ocupadas);
            sem_post(&mem->mutex_tarro);
        }
        else sem_post(&mem->despertar);
    }
}

void inicializar_memoria() {
    int id_shm = shmget((key_t) 1234,sizeof(struct memoria_compartida),IPC_CREAT | 0666);
    if ( id_shm == -1 ) {
        printf("Error al crear la memoria compartida.\n");
        exit(error_creacion_memoria);
    }

    void * p = shmat(id_shm,NULL,0);
    if ( p == (void*) -1) {
        printf("Error al insertar un dato en la memoria compartida.\n");
        exit(error_attach);
    }
    memoria_compartida mem = (memoria_compartida) p;
    sem_init(&mem->despertar,1,0);
    sem_init(&mem->mutex_tarro,1,1);
    sem_init(&mem->vacio,1,capacidad_tarro);
    mem->porciones_ocupadas = 0;
    if ( shmdt(p) ) {
        printf("Error en la desvinculación de la memoria.\n");
        exit(error_detach);
    }
}

void inicializar_oso() {
    int pid = fork();
    if ( !pid ) {
        int id_shm = shmget((key_t) 1234,sizeof(struct memoria_compartida),0666);
        if ( id_shm == -1 ) {
            printf("Error al vincularse a la memoria compartida.\n");
            exit(error_vinculacion_memoria);
        }
        void * p = shmat(id_shm,NULL,0);
        if ( p == (void*) -1) {
            printf("Error al insertar un dato en la memoria compartida.\n");
            exit(error_attach);
        }
        memoria_compartida mem = (memoria_compartida) p;
        oso(mem);
        exit(0);
    }
    else if ( pid == -1 ) {
```

```
        printf("Error en fork.\n");
        exit(error_fork);
    }
}

void inicializar_abejas() {
    for ( int  i = 0; i < cantidad_abejas; i++ ) {
        int pid = fork();
        int id_shm = shmget((key_t) 1234,sizeof(struct memoria_compartida),0666);
        if ( id_shm == -1 ) {
            printf("Error al vincularse a la memoria compartida.\n");
            exit(error_vinculacion_memoria);
        }
        void * p = shmat(id_shm,NULL,0);
        if ( p == (void*) -1) {
            printf("Error al insertar un dato en la memoria compartida.\n");
            exit(error_attach);
        }
        memoria_compartida mem = (memoria_compartida) p;
        if ( !pid ) {
            abeja(i,mem);
            exit(0);
        }
        else if ( pid == -1 ) {
            printf("Error en fork.\n");
            exit(error_fork);
        }
    }
}

int main() {
    inicializar_memoria();
    inicializar_oso();
    inicializar_abejas();
    for ( int i = 0; i < cantidad_abejas+1; i++ ) wait(NULL);
    return 0;
}
```

archivo_permisos.c:

```
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include "constantes.h"

mode_t mu = 0, mg = 0, mo = 0;

void asignar_mu(char * argv){
      int i, size = strlen(argv);
      for ( i=0; i<size; i++){
            if (argv[i] == 'r' || argv[i] == 'R') mu |= S_IRUSR;
            else if(argv[i] == 'w' || argv[i] == 'W') mu |= S_IWUSR;
            else if(argv[i] == 'x' || argv[i] == 'X') mu |= S_IXUSR;
            else {
                  printf("No existe tal modo de usuario.\n");
                  exit(error_modo);
            }
      }
}

void asignar_mg(char * argv){
      int i, size = strlen(argv);
      for ( i=0; i<size; i++){
            if (argv[i] == 'r' || argv[i] == 'R') mg |= S_IRGRP;
            else if(argv[i] == 'w' || argv[i] == 'W') mg |= S_IWGRP;
            else if(argv[i] == 'x' || argv[i] == 'X') mg |= S_IXGRP;
            else {
                  printf("No existe tal modo de grupo.\n");
                  exit(error_modo);
            }
      }
}
void asignar_mo(char * argv){
```

```
	int i, size = strlen(argv);
	for ( i=0; i<size; i++){
		if (argv[i] == 'r' || argv[i] == 'R') mo |= S_IROTH;
		else if(argv[i] == 'w' || argv[i] == 'W') mo |= S_IWOTH;
		else if(argv[i] == 'x' || argv[i] == 'X') mo |= S_IXOTH;
		else {
			printf("No existe tal modo de otros.\n");
			exit(error_modo);
		}
	}
}

int main(int argc, char * argv[]) {
	if ( argc < 5 ) {
		printf("Se deben especificar un archivo y tres permisos.\n");
		exit(error_parametros);
	}
	if ( strcmp(argv[2],"-") ) asignar_mu(argv[2]);
	if ( strcmp(argv[3],"-") ) asignar_mg(argv[3]);
	if ( strcmp(argv[4],"-") ) asignar_mo(argv[4]);
	if ( mu != 0 || mg != 0 || mo != 0 ) { // si no se asignó ningún permiso, no hacer nada
		if ( chmod(argv[1], mu | mg | mo) == -1 ) {
			int errnum = errno;
			if ( errnum == ENOENT ) {
				printf("No se pudo encontrar el archivo.\n");
				exit(error_archivo);
			}
			printf("Error al asignar los privilegios al archivo.\n");
			exit(error_asignacion);
		}
		printf("Permiso asignado correctamente.\n");
	}
	return 0;
}
```

manejador_memoria.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "constantes.h"

// el 9° bit de la TP es el de válido.
// el 17° bit del TLB es el de referenciada y el 18° el de válido.
char TP[256][9], TLB[16][18], frames_libres[256];
int peor_caso_TLB = 0;
FILE * arch_dir;

/*
 * si modo es 'r', busca en la TLB con numero_pag para obtener el frame asociado y
 * guardado en numero_frame. si el bit de válido asociado es 1, y la página es la
 * que se está buscando entonces retorna 1, sino 0.
 * si modo es 'w', busca en la TLB con numero_pag para luego escribir el valor de
 * numero_frame en la TLB. para este modo, el valor retornado no debe considerarse
 */
int entrada_TLB(char numero_pag[9],char numero_frame[9], char modo){
	int i=0, j, hit = 0;
	if ( modo == 'r' ) {
		while(!hit && i<16) {
			j = 0;
			hit = TLB[i][val] == '1';
			while ( hit && j < 8 ) {
				// seguir comparando mientras los bits en i sean iguales
				hit = numero_pag[j] == TLB[i][j];
				j++;
			}
			if ( !hit ) i++;
		}
		if ( hit ) {
			TLB[i][ref] = '1';  //como se encontró la página, se marca como referenciada
			for(j = 8; j < 16; j++) numero_frame[j-8] = TLB[i][j];
		}
	}
	else if ( modo == 'w' ) {
		i = 0;
```

```
int encontre_entrada = 0;
while ( i < ref && !encontre_entrada ) {
	if ( TLB[i][val] == '0' ) { // buscar entre las no-válidas
		encontre_entrada = 1;
		TLB[i][val] = '1';  //se valida la página.
		for ( j = 0; j < 8; j++ ) {
			TLB[i][j+8] = numero_frame[j];
			TLB[i][j] = numero_pag[j];
		}
	}
	i++;
}
if ( !encontre_entrada ) { // buscar entre las válidas, pero no referenciadas
	i = 0;
	while ( i < ref && !encontre_entrada ) {
		if ( TLB[i][ref] == '0' ) {
			encontre_entrada = 1;
			for ( j = 0; j < 8; j++ ) {
				TLB[i][j+8] = numero_frame[j];
				TLB[i][j] = numero_pag[j];
			}
		}
		i++;
	}
	if ( !encontre_entrada ) { // si todas fueron válidas o referenciadas
		for ( i = 0; i < 8; i++ ) {
			TLB[peor_caso_TLB][i+8] = numero_frame[i];
			TLB[peor_caso_TLB][i] = numero_pag[i];
		}
		TLB[peor_caso_TLB][ref] = '0';  // el bit de referenciado vuelve a 0
		peor_caso_TLB++;   // para que no se reemplace siempre la misma página.
		if ( peor_caso_TLB == 16 ) peor_caso_TLB = 0;  // para evitar segmentation fault.
	}
    }
    return hit;
}

/*
 * si modo es 'r', indexa en la TP con numero_pag para obtener el frame asociado y
 * guardado en numero_frame. si el bit de válido asociado es 1, retorna 1, sino 0.
 * si modo es 'w', indexa en la TP con numero_pag para luego escribir el valor de
 * numero_frame en la TP. para este modo, el valor retornado no debe considerarse
 */
int entrada_TP(char numero_pag[9], char numero_frame[9], char modo) {
    int i, indice_TP = 0, hit, j = 0;
    for ( i = 7; i > -1; i--) indice_TP += (numero_pag[i]-'0')*pot(10,j++);
    binario_decimal(&indice_TP);
    if ( modo == 'r' ) {
	if ( TP[indice_TP][8] == '1' ) { //bit de válido en 1
		hit = 1; // hit de TP
		for ( i = 0; i < 8; i++ ) numero_frame[i] = TP[indice_TP][i];
	}
	else hit = 0; // miss de TP
    }
    else if ( modo == 'w' ) { // colocar el numero_frame en numero_pag
	TP[indice_TP][8] = '1';
	for ( i = 0; i < 8; i++ ) TP[indice_TP][i] = numero_frame[i];
    }
    return hit;
}

/* Se busca un frame que este libre y se lo guarda en numero_frame.
 * Para ello, se mantiene una lista con todos los frames que estén libres.
 * Un frame libre se marca con un 0, mientras que uno ocupado se marca con un 1.
*/
void reservar_frame(char numero_frame[9]) {
    int i = 0, frame = -1;
    while ( i < 256 && frame == -1) {
	if ( frames_libres[i] == '0' ) {
		frame = i;
		frames_libres[i] = '1';
```

```
            }
            i++;
        }
        decimal_binario(frame,numero_frame);
}

/* a partir de direccion_logica, obtiene el numero de frame asociado, que se almacena
 * en numero_frame, accediendo al TLB o a la TP, como sea necesario.
 */
void traducir(char direccion_logica[17], char numero_frame[9]) {
        int i, hit;
        char numero_pag[8];
        for (i=0; i<8; i++) numero_pag[i]=direccion_logica[i];
        hit = entrada_TLB(numero_pag,numero_frame,'r');
        if ( hit ) printf("Hit en el TLB.\n");
        if ( !hit ) { //miss TLB
                printf("Miss en el TLB.\n");
                hit = entrada_TP(numero_pag,numero_frame,'r');
                if ( hit ) printf("Hit en la TP.\n");
                if ( !hit ) { //miss TP
                        printf("Miss en la TP.\n");
                        reservar_frame(numero_frame);
                        entrada_TP(numero_pag,numero_frame,'w');
                        printf("TP actualizada.\n");
                 }
                 entrada_TLB(numero_pag,numero_frame,'w');
                 printf("TLB actualizado.\n");
        }
}

// setea los valores de los parámetros en '0'
void inicializar_tablas() {
        int i, j;
        for ( i = 0; i < 256; i++) {
                frames_libres[i] = '0';
                for ( j = 0; j < 9; j++ ) TP[i][j] = '0';
        }
        for ( i = 0; i < 16; i++)
          for ( j = 0; j < 18; j++ ) TLB[i][j] = '0';
}

int main() {
        char offset[9], numero_frame[9], direccion_logica[17]; // 17 y 9 por el NULL al final de los string
        inicializar_tablas();
        arch_dir=fopen("memoria.txt","r");
        int i, j;
        if (arch_dir == NULL){
                printf("Error al abrir el archivo memoria.txt.\n");
                exit(error_archivo);
        }
        while ( fscanf(arch_dir, "%s", direccion_logica) != EOF ) {
                for ( i = 0; i < 16; i++ ) {
                        if ( direccion_logica[i] != '0' && direccion_logica[i] != '1' ) {
                                printf("Las direcciones ingresadas no son correctas.\n");
                                exit(error_direcciones);
                        }
                }
         if ( strlen(direccion_logica) != 16 ) {
                printf("Las direcciones ingresadas no son correctas.\n");
                exit(error_direcciones);
         }
         for( i = 0; i < 8; i++ ) offset[i] = direccion_logica[i+8];
         printf("Traducción de la dirección lógica %s:\n",direccion_logica);
                traducir(direccion_logica,numero_frame);
                printf("La dirección física es %s %s\n\n",numero_frame,offset);
        }
        fclose(arch_dir);
        printf("TLB (página marco)\n");
        for ( i = 0; i < 16; i++ ) {
         for ( j = 0; j < 8; j++ ) printf("%c", TLB[i][j]); // imprimir número página
         printf(" ");
         for ( j = 8; j < 16; j++ ) printf("%c", TLB[i][j]); // imprimir número de frame
```

```
        printf("\n");
      }

      printf("\n");
      printf("TP (marco)\n");
      for ( i = 0; i < 256; i++ ) {
            if ( TP[i][8] == '1' )  {
                  for ( j = 0; j < 8; j++ ) printf("%c", TP[i][j]); // imprimir numero de frame
            printf("\n");
       }
      }
      return 0;
}
```

constantes.h:

```
#define error_parametros -1
#define error_modo -2
#define error_asignacion -3
#define error_archivo -4
#define error_fork -5
#define error_send -6
#define error_receive -7
#define error_malloc -8
#define error_cola -9
#define error_creacion_memoria -10
#define error_pipe -11
#define error_leer_pipe -12
#define error_escribir_pipe -13
#define error_close_pipe -14
#define error_vinculacion_memoria -15
#define error_attach -16
#define error_detach -17
#define error_direcciones -18
#define ref 16
#define val 17

#define capacidad_tarro 10
#define cantidad_abejas 3
#define EA 1  #define AC 2   #define CD 3   #define DE 4  #define EB 5  #define BC 6   #define AB 7  #define EABC 8  #define ABCABC 9
#define ABCD 10

/*
 * conversión de numero_pag de binario a decimal
 */
void binario_decimal(int * numero) {
       int  bin = *numero, base = 1, resto;
       *numero = 0;
    while (bin > 0) {
        resto = bin % 10;
        *numero += resto * base;
        bin = bin / 10 ;
        base = base * 2;
    }
}

// calcula x elevado a la n
int pot(int x, int n) {
    int i, numero = 1;
    for (i = 0; i < n; i++) numero *= x;
    return numero;
}

// conversión de decimal a binario de la variable "decimal". el resultado, en binario, es guardado en la   * variable "binario".
void decimal_binario(int decimal, char binario[9]) {
      int i = 7;
      if ( decimal != 0 ) {
            while (decimal != 1) {
                  binario[i] = decimal % 2 + '0';
                  decimal /= 2;
                  i--;
            }
            if ( decimal == 1 ) binario[i--] = '1';
      }
      while ( i > -1 ) binario[i--] = '0';
}
```