**secuencia1\_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';

}