TEMA 3: Programación del sistema con C
Ejemplos

system.c

```c
#include <stdlib.h>

int main ()
{
        return system ("ls -l /");
}
```

fork0.c

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
        if (fork ())
                printf ("Este es el proceso padre\n");

        else
                printf ("Este es el proceso hijo\n");

        return 0;
}
```

fork1.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>


int main()
{
        pid_t hijo;
        printf ("PRINCIPAL: Mi PID es %d\n", (int) getpid ());

        hijo = fork();
        if (hijo){
                printf ("PADRE: Mi PID es %d y el de mi hijo es %d\n", getpid (), hijo);
                printf ("PADRE: Adios\n");
        }
        else {
                printf ("HIJO: Mi PID es %d y el de mi padre es %d\n",  getpid (),
                                getppid());
                sleep (10);     /* Esperamos 10 segundos */
        }

        return 0;
}
```

fork2.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>



int main()
{
        pid_t hijo;
        printf ("PRINCIPAL: Mi PID es %d\n", (int) getpid ());

        hijo = fork();
        if (hijo){
                printf ("PADRE: Mi PID es %d y el de mi hijo es %d\n", getpid (), hijo);
                wait (0);
                printf ("PADRE: Adios, mi hijo ha acabado\n");
        }
        else {
                printf ("HIJO: Mi PID es %d y el de mi padre es %d\n",  getpid (),
                                getppid());
                sleep (10);     /* Esperamos 10 segundos */
        }

        return 0;
}
```

fork3.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>



int main()
{
        pid_t hijo;
        int estado;
        printf ("PRINCIPAL: Mi PID es %d\n", (int) getpid ());

        hijo = fork();
        if (hijo){
                printf ("PADRE: Mi PID es %d y el de mi hijo es %d\n", getpid (), hijo);
                wait (&estado);
                printf ("PADRE: Adios, mi hijo ha acabado con estado %d\n", estado);
        }
        else {
                printf ("HIJO: Mi PID es %d y el de mi padre es %d\n",  getpid (),
                                getppid());
                sleep (10);     /* Esperamos 10 segundos */
        }

        return 0;
}
```

fork4.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int hijo (int numero){
        printf ("HIJO %d: Mi PID es %d\n", numero, getpid ());
        /* Ahora esperamos 20 segundos */
        sleep (20);
        return 0;
}



int main()
{
        pid_t hijo1, hijo2, hijo3;
        printf ("PRINCIPAL: Mi PID es %d\n", (int) getpid ());

        hijo1 = fork();
        if (hijo1){
                hijo2 = fork();
                if (hijo2){
                        hijo3 = fork();
                        if (hijo3){
                                printf ("PADRE: Mi PID es %d\n", getpid ());
                                printf ("PADRE: Adios\n");

                        }
                        else hijo (3);
                }
                else hijo (2);
        }
        else hijo (1);


        return 0;
}
```

fork5.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int hijo (int numero){
        printf ("HIJO %d: Mi PID es %d\n", numero, getpid ());
        /* Ahora esperamos 20 segundos */
        sleep (20);
        return 0;
}



int main()
{
        pid_t hijo1, hijo2, hijo3;
        int estado1, estado2, estado3;
        printf ("PRINCIPAL: Mi PID es %d\n", (int) getpid ());

        hijo1 = fork();
        if (hijo1){
                hijo2 = fork();
                if (hijo2){
                        hijo3 = fork();
                        if (hijo3){
                                printf ("PADRE: Mi PID es %d\n", getpid ());
                                waitpid (hijo1, &estado1, WUNTRACED);
                                waitpid (hijo2, &estado2, WUNTRACED);
                                waitpid (hijo3, &estado3, WUNTRACED);
                                printf ("PADRE: Adiós, mis hijos han acabado\n");
```

```
                                printf ("PADRE:       Hijo 1 acaba con estado %d\n",
                                    estado1);
                                printf ("PADRE:       Hijo 2 acaba con estado %d\n",
                                    estado2);
                                printf ("PADRE:       Hijo 3 acaba con estado %d\n",
                                    estado3);

                        }
                        else hijo (3);
                }
                else hijo (2);
        }
        else hijo (1);


        return 0;
}
```

exec.c

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

char *programa="ls";
char *argumentos[]={"ls", "-l", "/", NULL};

int main ()
{
        pid_t hijo;

        hijo = fork ();
        if (hijo != 0)
                /* Este es el proceso padre */
                return hijo;
        else {
                /* Ejecutamos PROGRAMA, buscando en el PATH */
                execvp (programa, argumentos);
                /* execvp vuelve solo si hay un error */
                fprintf (stderr, "Ha ocurrido un error en execvp\n");
                abort ();
        }
        return 0;
}
```

signal.c

```
/*
 *      Señales:
 *      Con SIGUSR1 hacemos que el programa ignore SIGINT
 *      Con SIGUSR2 hacemos que el programa responda ante SIGINT
 *
 */
#include <signal.h>
#include <stdio.h>

void f_sigusr1 (){
        fprintf (stderr, "Señal SIGUSR1 recibida\n");
        fprintf (stderr, "Ignorando SIGINT\n");
        signal (SIGINT, SIG_IGN);
}

void f_sigusr2 (){
        fprintf (stderr, "Señal SIGUSR2 recibida\n");
        fprintf (stderr, "Restaurando SIGINT\n");
        signal (SIGINT, SIG_DFL);
}
```

```c
int main (){
        signal (SIGUSR1, f_sigusr1);
        signal (SIGUSR2, f_sigusr2);
        while (1) pause ();
        return 0;
}
```

kill.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int hijo (int numero){
        printf ("HIJO %d: Mi PID es %d\n", numero, getpid ());
        /* Ahora esperamos 400 segundos */
        sleep (400);
        return 0;
}



int main()
{
        pid_t hijo1, hijo2, hijo3;
        printf ("PRINCIPAL: Mi PID es %d\n", (int) getpid ());

        hijo1 = fork();
        if (hijo1){
                hijo2 = fork();
                if (hijo2){
                        hijo3 = fork();
                        if (hijo3){
                                printf ("PADRE: Mi PID es %d\n", getpid ());
                                printf ("Tengo 3 hijos con los siguientes PID: \n");
                                printf ("1) %d\n", hijo1);
                                printf ("2) %d\n", hijo2);
                                printf ("3) %d\n", hijo3);
                                printf ("Matando hijo 1... Compruebalo\n"); kill (hijo1, SIGTERM);
                                sleep (5);
                                printf ("Matando hijo 2... Compruebalo\n"); kill (hijo2, SIGTERM);
                                sleep (5);
                                printf ("Matando hijo 3... Compruebalo\n"); kill (hijo3, SIGTERM);
                                sleep (5);
                                printf ("PADRE: Adios\n");

                        }
                        else hijo (3);
                }
                else hijo (2);
        }
        else hijo (1);


        return 0;
}
```

pipe0.c

```c
/*
 * COMUNICACION PADRE-HIJO A TRAVES DE TUBERIA
 *      fd[0]: Lectura
 *      fd[1]: Escritura
 *
 * Envio de datos PADRE->HIJO
 *      Cerrar fd[0], escribir en fd[1]
 *
 * Envio de datos HIJO->PADRE
 *      Cerrar fd[0], escribir en fd[1]
 *
 *
 */

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>


int main(void)
{
        int     fd[2], nbytes;
        pid_t   pid;
        char    mensaje[] = "Hola hijo!\n";
        char    buffer[80]="\0";
        char    buffer2[80]="\0";


        pipe(fd);
        if((pid = fork()) == -1)
        {
                /* No se ha podido crear proceso hijo */
                perror("fork");
                exit(1);
        }

        if(pid == 0)
        {
                /* Proceso hijo */
                /* Cierre del descriptor de salida en el hijo */
                close(fd[1]);

                /* Leer algo de la tuberia... el saludo! */
                nbytes = read(fd[0], buffer, sizeof(buffer));
                printf("HIJO: Cadena recibida: %s", buffer);
                close (fd[0]);
                exit(0);
        }
        else {
                /* Proceso padre */
                /* Cierre de descriptor de entrada en el padre */
                close(fd[0]);

                /* Enviar el saludo via descriptor de salida */
                write(fd[1], mensaje, strlen(mensaje));
                close(fd[1]);
        }

        return 0;
}
```

pipe1.c

```c
/*
 * COMUNICACION A TRAVES DE TUBERIA
 *      fd[0]: Lectura
 *      fd[1]: Escritura
 */

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
```

```c
int main(void)
{
        int     fd[2];
        pid_t   pid;
        char    buffer0[80];
        char    buffer1[80];
        char    buffer2[80];
        int     i;

        pipe(fd);


        if((pid = fork()) == 0)
        {
                /* HIJO1 */
                /* Envío de datos */
                do{
                                for (i=0; i<80; i++) buffer1[i]='\0';
                                fgets (buffer1, 80, stdin); printf (" HIJO1 -> HIJO2 \n");
                                write(fd[1], buffer1, strlen(buffer1));
                } while (strncmp(buffer1,"quit",4));
                exit (0);
        }

        if ((pid = fork())==0) {
                /* HIJO2 */
                /* Recepción de datos */
                do {
                        for (i=0; i<80; i++) buffer2[i]='\0';
                        read(fd[0], buffer2, sizeof(buffer2));
                        printf("HIJO2: %s", buffer2);
                } while (strncmp(buffer2, "quit",4));
                exit (0);
        }


        /* Proceso padre */
        /* Reenvío de datos */
        do {
                for (i=0; i<80; i++) buffer0[i]='\0';
                read(fd[0], buffer0, sizeof(buffer0));
                write(fd[1], buffer0, strlen(buffer0));
        } while (strncmp(buffer0, "quit",4));
        close (fd[0]);
        close(fd[1]);
        return 0;
}
```

fifo.c

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>

int main (){
        int i;
        char buffer[80];
//      umask (0);
        mkfifo ("f", 0777);


        FILE* fifo = fopen ("f", "w");
        for (;;){
                fgets (buffer, 79, stdin);
                fputs (buffer, fifo);
                fflush (fifo);
        }
        fclose (fifo);
        return 0;
}
```

hilos0.c

```c
/*      Para compilar:     gcc  -o  hilos0  hilos0.c  -lpthread  */

#include <stdio.h>
#include <pthread.h>

pthread_t  h1, h2;

void *hilo1 ();
void *hilo2 ();


int main() {
        pthread_create (&h1, NULL, hilo1, NULL);
        pthread_create (&h2, NULL, hilo2, NULL);
        return 0;

}



void *hilo1(){
        sleep (10);     /* Esperamos 10 segundos */
        pthread_exit(NULL);
}



void *hilo2(){
        sleep (10);     /* Esperamos 10 segundos */
        pthread_exit(NULL);
}
```

hilos1.c

```c
/*      Para compilar:     gcc  -o  hilos1  hilos1.c  -lpthread  */

#include <stdio.h>
#include <pthread.h>

pthread_t  h1, h2;

void *hilo1 ();
void *hilo2 ();


int main() {
        pthread_create (&h1, NULL, hilo1, NULL);
        pthread_create (&h2, NULL, hilo2, NULL);
        pthread_join (h1, NULL);
        pthread_join (h2, NULL);
        return 0;
}



void *hilo1(){
        printf ("HILO1: Mi TID es %d\n", pthread_self ());
        sleep (10);     /* Esperamos 10 segundos */
        pthread_exit(NULL);
}



void *hilo2(){
        printf ("HILO2: Mi TID es %d\n", pthread_self ());
        sleep (10);     /* Esperamos 10 segundos */
        pthread_exit(NULL);
}
```

sem0.c

```c
/*
 *      Programa PRODUCTOR-CONSUMIDOR
 *      Dada una cola circular compartida entre productor y consumidor
 *      se sincroniza su actualización mediante un semáforo s.
 *
 *      Graduar tiempos de productor y consumidor mediante usleep.
 *      Inicialmente la cola circular está llena.
 *      El productor muestra por pantalla los números que escribe en la cola
 *      en la primera columna.
 *      El consumidor muestra por pantalla los números que lee de la cola
 *      a partir de la segunda columna.
 *      Para finalizar el programa pulsar CTRL+C.
 *
 *      Para compilar:     gcc  -o  sem0  sem0.c  -lpthread
 */
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>     /* Para usleep, para graduar tiempos */

#define TAM 200

sem_t s;
typedef struct {
        int num[TAM];
        int inicio;
        int fin;
        enum {uno,dos} ultimo_avance;
} cola_circular;
cola_circular numeros;

void *productor ();
void *consumidor ();


int main(){
        int error1, error2;
        pthread_t  h1, h2;
        sem_init (&s, 0, 1);

        /* Inicializamos valores de la cola */
        for (numeros.inicio=0; numeros.inicio<TAM; numeros.inicio++)
               numeros.num[numeros.inicio]=numeros.inicio;
        numeros.inicio = 0;
        numeros.fin = 0;
        numeros.ultimo_avance = dos;   // Cola llena

        error1 = pthread_create (&h1, NULL, productor, NULL);
        error2 = pthread_create (&h2, NULL, consumidor, NULL);
        pthread_join (h1, NULL);
        pthread_join (h2, NULL);
        return 0;
}
```

```
void *productor () {
        int i=TAM;
        for (;;){
                sem_wait (&s);
                /* Si está lleno */
                if ((numeros.inicio==numeros.fin) && (numeros.ultimo_avance==dos))
                      sem_post (&s);
                /* Si no está lleno */
                else {
                        numeros.num[numeros.fin] = i;
                        printf ("\n%d", numeros.num[numeros.fin]);
                        numeros.fin = (numeros.fin + 1)%TAM;
                        numeros.ultimo_avance = dos;
                        i++;
                        sem_post (&s);
                        usleep (10000);
                }
        }
        pthread_exit(NULL);
}

void *consumidor () {
        for (;;) {
                sem_wait (&s);
                /* Si está vacio */
                if ((numeros.inicio==numeros.fin) && (numeros.ultimo_avance==uno))
                       sem_post (&s);
                /* Si no está vacio */
                else {
                        printf ("\t%d", numeros.num[numeros.inicio]);
                        numeros.inicio = (numeros.inicio + 1)%TAM;
                        numeros.ultimo_avance = uno;
                        sem_post (&s);
                        usleep (100);
                }
        }
        pthread_exit(NULL);
}
```

sem1.c

```c
/*
 * Suma los números impares entre 0 y 20, es decir
 * los números 1+3+5+7+9+11+13+15+17+19 = 100
 *
 * El primer hilo comprueba que el número es impar,
 * si lo es deja que el segundo hilo lo sume,
 * sino comprueba el siguiente número.
 *
 * Para compilar:     gcc  -o  sem1  sem1.c  -lpthread
 *
 */

#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>


sem_t s1, s2;
int num=0, suma=0;

void *p1 ();
void *p2 ();


int main(){
        int error1, error2;
        pthread_t  h1, h2;
        sem_init (&s2, 0, 0);
        sem_init (&s1, 0, 1);
        error1 = pthread_create (&h1, NULL, p1, NULL);
        error2 = pthread_create (&h2, NULL, p2, NULL);
        pthread_join (h1, NULL);
        pthread_join (h2, NULL);
        printf ("%s %d\n", "La suma es ->", suma);
        return 0;
}


void *p1(){
        int i;
        for (i=0; i<20; i++){
                sem_wait (&s1);
                printf ("HILO1: Número %d\n", i);
                if (i%2) {
                        num=i;
                        sem_post (&s2);
                }
                else sem_post (&s1);
        }
        pthread_exit(NULL);
}

void *p2(){
        int i;
        for (i=0; i<10; i++) {
                sem_wait (&s2);
                printf ("HILO2: Suma = %d + %d\n", suma, num);
                suma = suma+num;
                sem_post (&s1);
        }
        pthread_exit(NULL);
}
```

servidor0.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define PORT    3550                            /* El puerto que será abierto */
#define BACKLOG 2                               /* El número de conexiones en cola */

main()
{

   int fd1, fd2;                                /* Los descriptores de ficheros */
   struct sockaddr_in servidor, cliente;        /* Información del servidor y cliente */
   int size;                                    /* Para tamaño de sockaddr_in */

   /* Información del servidor en sockaddr_in */
   servidor.sin_family = AF_INET;
   servidor.sin_port = htons(PORT);             /* Es necesario utilizar htons */
   servidor.sin_addr.s_addr = INADDR_ANY;
      /* INADDR_ANY coloca nuestra dirección IP automáticamente */
   bzero(&(servidor.sin_zero),8);
      /* Escribimos ceros en el resto de la estructura */

   /* Conexión */
   fd1=socket(AF_INET, SOCK_STREAM, 0);
   bind(fd1,(struct sockaddr*)&servidor, sizeof(struct sockaddr));
   listen(fd1,BACKLOG);

   size=sizeof(struct sockaddr_in);
   while(1) {
      /* accept(): Aceptamos conexión */
      fd2 = accept(fd1,(struct sockaddr *)&cliente, &size);

      /* Mostramos IP del cliente y le enviamos un mensaje de bienvenida */
      printf("Se obtuvo una conexión desde %s\n", inet_ntoa(cliente.sin_addr) );
      send(fd2,"Bienvenido a mi servidor.\n",26,0);

      close(fd2); /* Cerramos fd2 */
   }
}
```

cliente0.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>           /* netbd.h es necesitada por la estructura hostent ;-) */


#define PORT 3550                       /* El Puerto Abierto del nodo remoto */
#define MAXDATASIZE 100                 /* El número máximo de datos en bytes */


int main(int argc, char *argv[])
{
   int numbytes;
   int fd;                              /* Descriptor de archivo */
   struct sockaddr_in servidor;         /* Información sobre la dirección del servidor */
   char buf[MAXDATASIZE];               /* Donde almacenaremos el texto recibido */
   struct hostent *he;
     /* Estructura que recibirá información sobre el nodo remoto */


   if (argc !=2) {
      printf("Uso: %s <Dirección IP>\n",argv[0]);
      exit(-1);
   }

   he=gethostbyname(argv[1]);                   /* llamada a gethostbyname() */

   /* Información del servidor en sockaddr_in */
   servidor.sin_family = AF_INET;
```

```
   servidor.sin_port = htons(PORT);            /* Es necesario utilizar htons */
   servidor.sin_addr = *((struct in_addr *)he->h_addr);
     /*he->h_addr pasa la información de ``*he'' a "h_addr" */
   bzero(&(servidor.sin_zero),8);
     /* Escribimos ceros en el resto de la estructura */

   /* Conexión */
   fd=socket(AF_INET, SOCK_STREAM, 0);
   connect(fd, (struct sockaddr *)&servidor, sizeof(struct sockaddr));
   numbytes=recv(fd,buf,MAXDATASIZE,0);
   buf[numbytes]='\0';

   /* Mostramos el mensaje recibido */
   printf("Mensaje del Servidor: %s\n",buf);

   close(fd);   /* Cerramos fd */

}
```

servidor1.c

```
/* Estos son los ficheros de cabecera usuales */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define PORT    3550                        /* El puerto que será abierto */
#define BACKLOG 2                           /* El número de conexiones en cola */

main()
{

   int fd1, fd2;                            /* Los descriptores de ficheros */
   struct sockaddr_in servidor, cliente;    /* Información del servidor y cliente */
   int size;                                /* Para tamaño de sockaddr_in */

   /* Información del servidor en sockaddr_in */
   servidor.sin_family = AF_INET;
   servidor.sin_port = htons(PORT);         /* Es necesario utilizar htons */
   servidor.sin_addr.s_addr = INADDR_ANY;   /* INADDR_ANY coloca nuestra dirección IP
automáticamente */
   bzero(&(servidor.sin_zero),8);           /* Escribimos ceros en el resto de la estructura */



   /* socket() */
   if ((fd1=socket(AF_INET, SOCK_STREAM, 0)) == -1 ) {
      printf("error en socket()\n");
      exit(-1);
   }

   /* bind() */
   if(bind(fd1,(struct sockaddr*)&servidor, sizeof(struct sockaddr))==-1) {
      printf("error en bind() \n");
      exit(-1);
   }

   /* listen() */
   if(listen(fd1,BACKLOG) == -1) {
      printf("error en listen()\n");
      exit(-1);
   }

   size=sizeof(struct sockaddr_in);
   while(1) {
      /* accept(): Aceptamos conexión */
      if ((fd2 = accept(fd1,(struct sockaddr *)&cliente, &size))==-1) {
         printf("error en accept()\n");
         exit(-1);
      }

      /* Mostramos IP del cliente y le enviamos un mensaje de bienvenida */
```

```
      printf("Se obtuvo una conexión desde %s\n", inet_ntoa(cliente.sin_addr) );
      send(fd2,"Bienvenido a mi servidor.\n",26,0);

      close(fd2); /* Cerramos fd2 */
   }
}
```

cliente1.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>                      /* netbd.h es necesitada por la estructura hostent ;-) */


#define PORT 3551                       /* El Puerto Abierto del nodo remoto */
#define MAXDATASIZE 100                 /* El número máximo de datos en bytes */


int main(int argc, char *argv[])
{
   int numbytes;
   int fd;                              /* Descriptor de archivo */
   struct sockaddr_in servidor;         /* Información sobre la dirección del servidor */
   char buf[MAXDATASIZE];               /* Donde almacenaremos el texto recibido */
   struct hostent *he;                  /* Estructura que recibirá información sobre el nodo remoto
*/


   if (argc !=2) {                      /* esto es porque nuestro programa necesitará la IP */
      printf("Uso: %s <Dirección IP>\n",argv[0]);
      exit(-1);
   }

   if ((he=gethostbyname(argv[1]))==NULL){      /* llamada a gethostbyname() */
      printf("gethostbyname() error\n");
      exit(-1);
   }

   /* Información del servidor en sockaddr_in */
   servidor.sin_family = AF_INET;
   servidor.sin_port = htons(PORT);             /* Es necesario utilizar htons */
   servidor.sin_addr = *((struct in_addr *)he->h_addr);
                                    /*he->h_addr pasa la información de ``*he'' a "h_addr" */
   bzero(&(servidor.sin_zero),8);               /* Escribimos ceros en el resto de la estructura */


   /* socket() */
   if ((fd=socket(AF_INET, SOCK_STREAM, 0))==-1){
      printf("socket() error\n");
      exit(-1);
   }

   /* connect() */
   if(connect(fd, (struct sockaddr *)&servidor, sizeof(struct sockaddr))==-1){
      printf("connect() error\n");
      exit(-1);
   }

   /* recv() */
   if ((numbytes=recv(fd,buf,MAXDATASIZE,0)) == -1){
      printf("Error en recv() \n");
      exit(-1);
   }
   buf[numbytes]='\0';

   /* Mostramos el mensaje recibido */
   printf("Mensaje del Servidor: %s\n",buf);

   close(fd);   /* Cerramos fd */

}
```