Exemples de programació amb Linux

Carles Pina i Estany (cpina@salleurl.edu)
Juny 2004

Índex de contingut

Programació amb processos Unix

Introducció

Aquí tenim un recull de una sèrie d'exemples de programació de processos i intercomunicació entre ells amb Unix.

Podreu trobar els exemples en format digital així com aquest document a http://pinux.info/lasalle/tt

La bibliografia per aquest tema es podria basar amb el llibre "Advanced Programming in the UNIX Environment", de "W. Richard Stevens" Editorial Addislon Wesley, disponible a la biblioteca de la Salle.

Veurem com crear processos, com enviar senyals entre ells, com comunicar-los (senyals, pipes, memòria compartida, missatges, etc.)

01-fork.c

```
 1      #include <stdio.h>
 2      #include <stdlib.h>
 3      #include <unistd.h>
 4      #include <sys/types.h>
 5      #include <sys/wait.h>
 6
 7      /*Carles Pina i Estany  -  carles@pinux.info
 8       * Febrer 2004*/
 9
10
11      /*Exemple fork*/
12
13      void fill ();
14
15      int main () {
16            int status;
17            pid_t pid;
18
19            pid=fork();
20
21            if (pid==0) {
22                  /*Sóc el fill*/
23                  fill ();
24            }
25            /*Sóc el pare*/
26
27            printf("Jo sóc el pare, i vaig a esperar al fill\n");
28
29            status=0;
30            wait(&status);
31
32            printf("Sóc el pare, adeu!\n");
33            return (0);
34      }
35
36      void fill () {
37            printf("Sóc el fill\n");
```

```
38            sleep(1);
39            printf("Sóc el fill, segona part\n");
40            exit(0);
41
42      }
43
```

02-execl.c

```
 1      #include <stdio.h>
 2      #include <stdlib.h>
 3      #include <unistd.h>
 4
 5      /* Carles Pina i Estany - carles@pinux.info
 6       * Febrer 2004*/
 7
 8      /*Exemple de execl*/
 9
10      int main () {
11            printf("Hola bon dia\n");
12
13
14            execl("/bin/ls","ls",(char *)0);
15            printf("Adeu\n");
16
17            return (0);
18
19      }
20
```

03-fork.c

```
 1      #include <stdio.h>
 2      #include <stdlib.h>
 3      #include <sys/types.h>
 4      #include <unistd.h>
 5      #include <sys/wait.h>
 6
 7      /*Carles Pina i Estany  -  carles@pinux.info
 8       * Febrer 2004*/
 9
10      /*Fork, veure direccions "locals" i contingut diferent*/
11
12
13      int main () {
14            int pid;
15            int a;
16            a=3;
17            printf("Direccio de a: %x\n",(unsigned int)&a);
18
19            pid=fork();
20
21            if (pid>0) {
```

```
22                /*Estic al pare*/
23                a=5;
24                printf("a del pare: %x. Val: %d\n",(unsigned int)&a,a);
25                wait(0);
26          }
27          else {
28                sleep(1);
29                printf("a del fill: %x. Val: %d\n",(unsigned int)&a,a);
30          }
31          return (0);
32    }
```

04-execl.c

```
 1    #include <stdio.h>
 2    #include <stdlib.h>
 3    #include <unistd.h>
 4    #include <sys/types.h>
 5    #include <sys/wait.h>
 6
 7    /* Carles Pina i Estany - carles@pinux.info
 8     * Febrer 2004*/
 9
10    /*Exemple de execl, i esperant pel fill*/
11
12    int main () {
13          pid_t pid;
14
15          printf("Hola bon dia\n");
16
17          pid=fork();
18          if (pid>0) {
19                //Sóc el pare
20                waitpid(pid,(int*)0,0);
21                printf("Ja he acabat d'executar\n");
22          }
23          else {
24                //sóc el fill
25                execl("/bin/ls","ls",(char *)0);
26          }
27          printf("Adeu!\n");
28          return (0);
29
30    }
31
```

05-pipe.c

```
 1      #include <stdio.h>
 2      #include <stdlib.h>
 3      #include <unistd.h>
 4      #include <sys/types.h>
 5      #include <sys/wait.h>
 6
 7
 8      #define LLEGIR 0
 9      #define ESCRIU 1
10
11      /*Carles Pina i Estany  -  carles@pinux.info
12       * Febrer 2004*/
13
14
15      /*exempe PIPE*/
16
17      void fill (int p[2]);
18
19
20      int main () {
21            int p[2];
22            pid_t pid;
23            char buf[100];
24            int status;
25            int ret;
26
27            pipe(p);
28
29            pid=fork();
30
31            if (pid==0) {
32                  /*Sóc el fill*/
33                  fill (p);
34            }
35            /*Sóc el pare*/
36            close (p[LLEGIR]);
37
38            strcpy(buf,"Hola Bon Dia\n");
39
40            printf("P: Vaig a escriure a fill\n");
41            ret=write (p[ESCRIU],buf,strlen(buf)+1);
42
43            status=0;
44            printf("P: Estic esperant a fill\n");
45            wait (&status);
46
47            return (0);
48      }
49
50      void fill (int p[2]) {
51            char buf[100];
52
53            close (p[ESCRIU]);
54
```

```
55            read (p[LLEGIR],buf,sizeof(buf));
56            printf("F: he llegit: %s\n",buf);
57
58            exit(0);
59      }
60
```

06-pipe.c

```
 1      #include <stdio.h>
 2      #include <stdlib.h>
 3      #include <unistd.h>
 4      #include <sys/types.h>
 5      #include <sys/wait.h>
 6      #include <errno.h>
 7
 8      #define LLEGIR 0
 9      #define ESCRIU 1
10
11      /*Carles Pina i Estany  -  carles@pinux.info
12       * Febrer 2004*/
13
14
15      /*exempe PIPE -AMB UN "ERROR"*/
16
17      void fill (int p[2]);
18
19
20      int main () {
21            int p[2];
22            pid_t pid;
23            char buf[100];
24            int status;
25            int ret;
26
27            pipe(p);
28
29            pid=fork();
30
31            if (pid==0) {
32                  /*Sóc el fill*/
33                  fill (p);
34            }
35            /*Sóc el pare*/
36            close (p[LLEGIR]);
37
38            strcpy(buf,"Hola Bon Dia\n");
39
40            printf("P: Vaig a escriure a fill\n");
41            ret=write (p[ESCRIU],buf,strlen(buf)+1);
42
43            strcpy(buf,"Hola Bon Dia2\n");
44            printf("P: Vaig a escriure a fill\n");
45            ret=write (p[ESCRIU],buf,strlen(buf)+1);
```

```
46
47
48            status=0;
49            printf("P: Estic esperant a fill\n");
50            wait (&status);
51
52            return (0);
53      }
54
55      void fill (int p[2]) {
56            char buf[100];
57            char *seg;
58
59            close (p[ESCRIU]);
60
61            read(p[LLEGIR],buf,sizeof(buf));
62            printf("F: he llegit: %s\n",buf);
63
64            read(p[LLEGIR],buf,sizeof(buf));
65            printf("F: he llegit: %s\n",seg);
66
67            exit(0);
68      }
69
```

07-signal.c

```
 1      #include <signal.h>
 2      #include <stdio.h>
 3      #include <stdlib.h>
 4      #include <sys/types.h>
 5      #include <unistd.h>
 6
 7      /*Carles Pina i Estany  -  carles@pinux.info
 8       * Febrer 2004*/
 9
10
11      void hola(int i);
12
13      int main () {
14            int i=0;
15
16            signal(SIGUSR1,hola);
17
18            for (i=0;;i++) {
19                  printf("bla bla bla: %d\n",i);
20                  kill (getpid(),SIGUSR1);
21                  sleep(1);
22            }
23            return(0);
24      }
25
26      void hola (int i) {
27            printf("EXECUTO HOLA. i:%d.\n",i);
28      }
29
```

08-signal.c

```
 1      #include <signal.h>
 2      #include <stdio.h>
 3      #include <unistd.h>
 4
 5
 6
 7      /*Carles Pina i Estany  -  carles@pinux.info
 8       * Febrer 2004*/
 9
10
11      void controlc ();
12      void signal_term();
13
14      int main() {
15        signal(SIGINT,controlc);
16        signal(15,signal_term);
17
18        for (;;) {
19              printf("Ping!\n");
20              sleep(1);
21        }
22
23
24        printf("Sortida correcte\n");
25        return (0);
26      }
27
28      void controlc () {
29            printf("Has apretat Control+c\n");
30      }
31
32      void signal_term() {
33            printf("Rebuda senyal de morir. Però passo...\n");
34      }
```

09-signal.c

```
 1      #include <signal.h>
 2      #include <stdio.h>
 3      #include <stdlib.h>
 4      #include <sys/types.h>
 5      #include <unistd.h>
 6
 7      /*Carles Pina i Estany  -  carles@pinux.info
 8       * Febrer 2004*/
 9
10
11      static void senyal_fill ();
12      static void senyal_pare ();
13
14      int main () {
15            pid_t pid;
```

```
16
17            pid=fork();
18
19            if (pid>0) {
20                  //Sóc el pare
21                  for (;;) {
22                        signal(SIGUSR2,senyal_pare);
23                        kill (pid,SIGUSR1);
24                        kill (getpid(),SIGUSR2);
25                        sleep(1);
26                  }
27            }
28            else {
29                  //Sóc el fill
30                  signal(SIGUSR1,senyal_fill);
31                  for (;;) {
32                        printf("Espero una senyal...\n");
33                        pause();
34                  }
35            }
36      }
37
38      static void senyal_fill () {
39            printf("Sóc la senyal del fill\n");
40      }
41
42      static void senyal_pare () {
43            printf("Sóc la senyal del pare\n");
44      }
```

10-alarm.c

```
 1      #include <signal.h>
 2      #include <stdio.h>
 3      #include <stdlib.h>
 4      #include <sys/types.h>
 5      #include <unistd.h>
 6
 7      /*Carles Pina i Estany  -  carles@pinux.info
 8       * Febrer 2004*/
 9
10
11      static void senyal_alarma();
12
13      int main () {
14            char buffer[101];
15
16            signal (SIGALRM,senyal_alarma);
17            alarm(2);
18
19            for (;;) {
20                  printf("Digues alguna cosa: ");
21                  fgets(buffer, sizeof(buffer),stdin);
22                  printf("Has dit: %s\n",buffer);
```

```
23
24            }
25            return (0);
26      }
27
28      static void senyal_alarma () {
29            printf("ALARMA!!!!\n");
30            signal (SIGALRM,senyal_alarma);
31      }
```

11-memoria.c

```
 1      #include <stdio.h>
 2      #include <stdlib.h>
 3      #include <unistd.h>
 4      #include <sys/types.h>
 5      #include <sys/ipc.h>
 6      #include <sys/shm.h>
 7
 8      #define MEM 568
 9
10      /*Carles Pina i Estany  -  carles@pinux.info
11       * Febrer 2004*/
12
13
14      /*Exemple de compartició de memòria*/
15      /*Fa un pare i 5 fills i comparteixen una regió de memòria*/
16
17      void fill ();
18
19      int main () {
20            int shmid;
21            int *shmptr;
22            pid_t pid;
23            int t;
24
25            /*Aquí demanarà 100 bytes de memòria, els fa, dona els permisos
26             * de lectura per tothom i posa que la clau és "10" */
27
28            shmid=shmget((key_t)10, MEM, IPC_CREAT|0666);
29            perror("error");
30            printf("Tinc el shmid: %d\n",shmid);
31
32            /*enllaça la regió de memòria a un punter normal i corrent*/
33            shmptr=shmat(shmid,0,0);
34
35            /*poso dades*/
36            for (t=0;t<50;t++) {
37                  shmptr[t]=2*t;
38            }
39
40
41            for (t=0;t<4;t++) {
42                  pid=fork();
```

```
43                if (pid==0) {
44                      /*Som un fill*/
45                      fill ();
46                }
47                sleep(1);
48                /*El primer fill veurà que shmptr[0] és 0, però
49                 * els altres ja veuran 25!!!*/
50                shmptr[0]=25;
51          }
52
53          sleep (15);
54          printf("Sóc el pare. Ara allibero memòria\n");
55
56
57          shmdt(shmptr);    /*Des-junto*/
58
59          shmctl(shmid, IPC_RMID,0); /*Allibero el el recurs!!!!*/
60
61          /* amb ipcs i ipcrm poden llistar i eliminar regions
62           * de memòria compartida que s'han quedat penjades...*/
63
64          return (0);
65    }
66
67    void fill() {
68          int *ptr;
69          int ident,t;
70
71          printf("Sóc un nou fill\n");
72          sleep(2);
73          /*Busequem l'identificador*/
74          ident=shmget((key_t) 10, MEM, 0666);
75          printf("Soc el fill, tinc el shmid: %d\n",ident);
76
77          /*fem que ptr apunti al començament de l'identificador*/
78          ptr=(int*)shmat(ident, NULL, 0);
79
80          for (t=0;t<5;t++) {
81                /*Si és 25 fem que sigui 1000...*/
82                if (ptr[0]==25) ptr[0]=1000;
83                printf("Sóc el fill. A la posició %d hi ha: %d\n",t,ptr
[t]);
84
85                sleep(1);
86          }
87
88          printf("Fi del fill\n");
89          exit(0);
90                      /*Forcem que acabi el fill
91                        pq. sinó torna a entrar al bucle!*/
92    }
93
```

12-semafors.c

```
 1      #include <sys/types.h>
 2      #include <sys/ipc.h>
 3      #include <sys/sem.h>
 4      #include <stdio.h>
 5      #include <unistd.h>
 6      #include <signal.h>
 7
 8
 9      /* Carles Pina i Estany - carles@pinux.info
10       * Febrer 2004 */
11
12      /*http://66.102.11.104/search?q=cache:UpdbPwN-
OM0J:www.infor.uva.es/~benja/semaforos-memoria.html+ejemplo+sem%C3%
A1foros+unix&hl=en&ie=UTF-8*/
13
14      int inicia (int valor);
15      void P (int semaforo);
16      void V (int semaforo);
17      int semcall (int semaforo, int operacion);
18
19      int inicia (int valor) {
20            int id;
21
22            union semun {
23                  int val;
24                  struct semid_ds *buf;
25                  ushort *array;
26            } arg;
27
28            if ((id=semget(IPC_PRIVATE, 1, (IPC_CREAT|0666))) == -1) {
29                  perror("Error al crear el semáforo.");
30                  return(-1);
31            }
32
33            arg.val = valor;
34            if (semctl(id, 0, SETVAL, arg) == -1) {
35                  perror("Error al inicializar el semáforo.");
36                  return (-1); /*error en inicializacion*/
37            }
38            return (id);
39      }
40
41      void P (int semaforo) {
42            if ( semcall(semaforo, -1) == -1 ) {
43                  perror("Error en operación P.");
44            }
45
46      }
47
48      void V (int semaforo) {
49            if ( semcall(semaforo, 1) == -1 ) {
```

```
50                perror("Error en operación V.");
51          }
52    }
53
54    int semcall (int semaforo, int operacion) {
55          struct sembuf sb;
56          sb.sem_num = 0;
57          sb.sem_op = operacion;
58          sb.sem_flg = 0;
59
60          return ( semop(semaforo, &sb, 1) ); /*devuelve -1 si error */
61    }
62
63    int main () {
64          pid_t pid;
65          int mutex;
66
67          mutex=inicia(1);
68
69          pid=fork();
70
71          if (pid>0) {
72                //Sóc el pare
73                sleep(1);
74                P(mutex); //bloqueja
75                printf("Ara espero 1 segon\n");
76                sleep(1);
77                V(mutex); //desbloqueja
78
79                sleep(2);
80                kill(pid,15);
81          }
82          else {
83                //Sóc el fill
84                for (;;) {
85                      P(mutex);
86                      fprintf(stderr,".");
87                      V(mutex);
88                }
89          }
90
91          return (0);
92    }
```

13-missatges.c

```
 1      #include <stdio.h>
 2      #include <stdlib.h>
 3      #include <sys/types.h>
 4      #include <sys/ipc.h>
 5      #include <sys/msg.h>
 6      #include <unistd.h>
 7      #include <sys/types.h>
 8      #include <sys/wait.h>
 9
10      struct missatge {
11            long tipus;
12            char dades[1024];
13      };
14
15      void fill (int ident);
16
17      int main () {
18            pid_t pid;
19            struct missatge m;
20            int a=0;
21            int ident;
22            m.tipus=1;
23            strcpy(m.dades,"hola bon dia");
24
25            ident=msgget((key_t) 12, IPC_CREAT|0600);
26
27            pid=fork();
28
29            if (pid>0) {
30                  /*Sóc el pare*/
31                  msgsnd(ident, &m, sizeof(m), 0);
32                  wait(&a);
33                  /*esperem el fill i així segur que tindrem missatges*/
34                  msgrcv(ident,&m,sizeof(m),1,0);
35                  printf("Pare. Missatge rebut: %s\n",m.dades);
36            }
37            else {
38                  /*Sóc el fill*/
39                  printf("hola\n");
40                  fill(ident);
41            }
42            msgctl(ident,IPC_RMID,0);
43            return (0);
44      }
45
46      void fill (int ident) {
47            struct missatge m;
48            printf("Estic al fill\n");
49            sleep(1);
50            printf("Després sleep\n");
51            printf("ident: %d\n",ident);
52            msgrcv(ident, &m, sizeof(m), 1, 0);
53
54            printf("Sóc el fill. He rebut un missatge: %s\n",m.dades);
```

```
55            strcpy(m.dades, "missatge rebut!!!");
56            msgsnd(ident,&m,sizeof(m),0);
57            exit(0);
58      }
```

14-socket_client.c

```
 1      #include <stdio.h>
 2      #include <netinet/in.h>
 3      #include <netdb.h>
 4      #include <unistd.h>
 5
 6      /* Carles Pina i Estany - carles@pinux.info
 7       * Febrer 2004*/
 8
 9      /* Es connecta a un servidor Web i agafa una pàgina Web */
10
11      /*PROTOCOL HTTP:
12       *
13       * GET /index.php HTTP/1.1
14       * Host: www.pinux.info:80
15
16       */
17
18      #define SERVIDOR_I_PORT "pinux.info:80"
19      #define SERVIDOR "pinux.info"
20      #define PAGINA_WEB "/prova.html"
21
22      int main () {
23            int fd;
24            struct sockaddr_in desti;
25            struct hostent *he;
26            char buffer[100];
27            int llegit;
28
29            fd=socket(AF_INET,SOCK_STREAM,0);
30
31            he=gethostbyname("pinux.info");
32            desti.sin_family=AF_INET;
33            desti.sin_port=htons(80);
34
35            desti.sin_addr=*((struct in_addr*)he->h_addr);
36
37            connect(fd,(struct sockaddr*)&desti,sizeof(struct sockaddr));
38
39            //podem fer servir el fd de forma "normal"
40
41            snprintf(buffer,sizeof(buffer),"GET /prova.html HTTP/1.1\n");
42            write (fd,buffer,strlen(buffer));
43
44            snprintf(buffer,sizeof(buffer),"Connection: close\n");
45            write (fd,buffer,strlen(buffer));
46
47
```

```
48            snprintf(buffer,sizeof(buffer),"Host: www.pinux.info:80\n\n");
49            write (fd,buffer,strlen(buffer));
50
51            printf("Començo a llegir\n");
52
53            while ((llegit=read(fd,buffer,sizeof(buffer)))>0)
54                  write (1,buffer,llegit);
55
56            close(fd);
57            return(0);
58      }
```

15-socket_servidor.c

```
 1      #include <stdio.h>
 2      #include <netinet/in.h>
 3      #include <sys/types.h>
 4      #include <sys/wait.h>
 5      #include <sys/socket.h>
 6      #include <arpa/inet.h>
 7      #include <netdb.h>
 8      #include <unistd.h>
 9      #include <strings.h>
10
11
12      /* Carles Pina i Estany - carles@pinux.info
13       * Febrer 2004*/
14
15      void atendre (int newfd);
16
17      int main () {
18            struct sockaddr_in servaddr;
19            char jo[100];
20            int sockfd,newfd;
21            struct sockaddr_in client_info;
22            struct hostent *host;
23            struct in_addr *ip;
24            int port=7879;
25            int pid_recullit;
26
27            int temp;
28            char nom_temp[100];
29            pid_t pid;
30            int addrlen;
31
32            sockfd=socket(AF_INET,SOCK_STREAM,0);
33
34            bzero(&servaddr,sizeof(servaddr));
35            servaddr.sin_family=AF_INET;
36
37            gethostname(nom_temp,100);
38
39            host=gethostbyname(nom_temp);
40            ip=(struct in_addr*)host->h_addr;
```

```
    41            snprintf(jo,16,"%s",inet_ntoa(*ip));
    42            printf("%s\n",inet_ntoa(*ip));
    43
    44            servaddr.sin_port=htons(port);
    45
    46            temp=bind(sockfd,(struct sockaddr*)&servaddr,sizeof(struct
sockaddr_in));
    47            if (temp!=0) {
    48                  fprintf(stderr,"Bind fracassat\n");
    49                  return (1);
    50            }
    51            listen (sockfd,10);
    52            addrlen=sizeof(struct sockaddr_in);
    53
    54            for (;;) {
    55                  bzero(&client_info,sizeof(client_info));
    56                  printf("Esperant connexio\n");
    57                  newfd=accept(sockfd,(struct sockaddr*)
&client_info,&addrlen);
    58                  fprintf(stderr,"Connectat client. IP: %s\n",inet_ntoa
(client_info.sin_addr));
    59
    60                  pid=fork();
    61                  if (pid==0) {
    62                        //Sóc el fill
    63                        close(sockfd);
    64                        atendre(newfd);
    65                  }
    66                  close(newfd); //el pare no el necessita
    67                  printf("netejo fills\n");
    68                  pid_recullit=waitpid(-1,NULL,WNOHANG);
    69                  printf("He recullit el pid: %d\n",pid_recullit);
    70            }
    71            return(0);
    72      }
    73
    74      void atendre (int newfd) {
    75            char buffer[100];
    76            char temporal[100];
    77            int llegit;
    78
    79            printf("Ja estic atenent al fill\n");
    80
    81            snprintf(temporal,100,"Introdueix texte\n");
    82            write(newfd,temporal,strlen(temporal));
    83
    84            llegit=read (newfd,buffer,sizeof(buffer));
    85
    86            buffer[llegit]='\0';
    87
    88            snprintf(temporal,100,"Has dit: %s\n",buffer);
    89            printf("Diré a client: %s\n",temporal);
    90
    91            write(newfd,temporal,strlen(temporal));
    92
```

```
93              close(newfd);
94              exit(0);
95      }
```

16-comandes.c

```
 1      #include <stdio.h>
 2      #include <stdlib.h>
 3      #include <sys/types.h>
 4      #include <unistd.h>
 5
 6      /*Carles Pina i Estany - carles@pinux.info
 7       * Febrer 2004*/
 8
 9      //Petita "simulació" de un interpret de comandes
10      //faltaria afegir-hi més comandes, opcions, redireccions, etc.
11
12      int main () {
13            int opcio;
14            int pid;
15
16            printf("EL MINI-SHELL\n");
17            printf("1: ls ||  2: df || 3: exit\n");
18
19            while(opcio!=3) {
20                  scanf("%d",&opcio);
21
22                  printf("Has seleccionat: %d\n",opcio);
23
24                  switch(opcio) {
25                        case 1:
26                        pid=fork();
27                        if (pid==0) {
28                              //Sóc el fill
29                              execl("/bin/ls","/bin/ls",NULL);
30                        }
31                        waitpid(&pid);
32                        break;
33
34                        case 2:
35                        pid=fork();
36                        if (pid==0) {
37                              //Sóc el fill
38                              execl("/usr/bin/du","/usr/bin/ls",NULL);
39                        }
40                        waitpid(&pid);
41                        break;
42                  }
43            }
44            return (0);
45      }
```

Programació amb Threads POSIX

Introducció

En aquest altre capítol veurem un sistema alternatiu de creació i sincronització de processos alternatiu al sistema de Unix.

Serà mitjançant les llibreries pthreads (POSIX Threads).

Bibliografia: “Programming with POSIX Threads”, de “David R. Butenhof”, disponible també a la biblioteca de la Salle.

Els conceptes en general seran semblants a Programació Tradicional de Unix (sincronització, creació de processos, etc.), tot i que la sintaxis serà diferent i evidentment tindrà algunes avantatges (més lleugeresa al crear threads i conmutar entre ells) i algunes desavantatges (necessàries les llibreries, etc.)

Destacar que aquest tema no substitueix del tot a l'anterior: amb Threads POSIX no disposem de Sockets, FIFOs, etc.

01-creacio.c

```
 1      #include <stdio.h>
 2      #include <stdlib.h>
 3      #include <pthread.h>
 4      #include <unistd.h>
 5
 6      /*Carles Pina i Estany  -  carles@pinux.info
 7       * Febrer 2004*/
 8
 9      /*Fem un fill, i esperem a que acabi -join*/
10
11      void *funcio(void *parametre) {
12            printf("Sóc el fill\n");
13            sleep(1);
14            printf("Sóc el fill i ja he esperat 1 seg\n");
15            pthread_exit(NULL);
16      }
17
18      int main() {
19            pthread_t fil;
20            pthread_attr_t attr;
21            int i;
22            printf("Sóc el pare abans de fer el fill\n");
23
24            pthread_attr_init( &attr );
25            pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
26
27            pthread_create(&fil, &attr, funcio,(void*)&i);
28
29            printf("Sóc el pare\n");
30
31            pthread_join(fil,NULL);
32            printf("Sóc el pare després d'esperar al fill\n");
33
34            return(0);
35      }
```

02-passparam.c

```
 1      #include <stdio.h>
 2      #include <stdlib.h>
 3      #include <pthread.h>
 4
 5      /*Carles Pina i Estany  -  carles@pinux.info
 6       * Febrer 2004*/
 7
 8
 9      /*Fem un fill, però li passem un valor. Esperem a que acabi*/
10
11      void *funcio(void *parametre) {
12            int valor;
13
14            valor=*((int*)parametre);
15
16            printf("Sóc el fill i m'han passat: %d\n",valor);
17            pthread_exit(NULL);
18      }
19
20      int main() {
21            pthread_t fil;
22            pthread_attr_t attr;
23            int i;
24
25            i=27;
26
27            printf("Sóc el pare abans de fer el fill\n");
28
29            pthread_attr_init( &attr );
30            pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
31
32            pthread_create(&fil, &attr, funcio,(void*)&i);
33
34            pthread_join(fil,NULL);
35            printf("Sóc el pare després d'esperar el fill\n");
36
37            return(0);
38      }
```

03-passparam2.c

```
 1      #include <stdio.h>
 2      #include <stdlib.h>
 3      #include <pthread.h>
 4
 5      /*Carles Pina i Estany  -  carles@pinux.info
 6       * Febrer 2004*/
 7
 8
 9      /*Fem un fill, i el fill ens retorna un valor*/
10
11      int retorna=25;
```

```
12
13      void *funcio(void *parametre) {
14            int a;
15            a=21;
16
17            pthread_exit((void*)a);
18      //    return((void*)23);
19      }
20
21      int main() {
22            pthread_t fil;
23            pthread_attr_t attr;
24            int i;
25            int resultat;
26
27            i=27;
28
29            printf("Sóc el pare abans de fer el fill\n");
30
31            pthread_attr_init( &attr );
32            pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
33
34            pthread_create(&fil, NULL, funcio,NULL);
35
36            pthread_join(fil,(void**)&resultat);
37
38            printf("Sóc el pare després d'esperar el fill\n");
39            printf("Resultat: %d\n",resultat);
40
41            return(0);
42      }
```

04-cancel.c

```
 1      #include <stdio.h>
 2      #include <pthread.h>
 3      #include <unistd.h>
 4
 5      /*Carles Pina i Estany  -  carles@pinux.info
 6       * Febrer 2004*/
 7
 8
 9      /*Exemple de cancelació de fill*/
10
11      void *codi (void *arg) {
12            int i;
13
14            for (i=0;i<5;i++) {
15                  printf("Sóc el fill\n");
16                  sleep(1);
17            }
18
19            return (NULL);
20      }
```

```
21
22      int main () {
23            pthread_t thread;
24            int i;
25
26            pthread_create(&thread,NULL,&codi,NULL);
27
28            sleep (2);
29
30            printf("Sóc el thread pare, vaig a fer cancel a fill...\n");
31
32            pthread_cancel(thread);
33
34            printf("Fet\n");
35
36            for (i=0;i<2;i++) {
37                  printf("Sóc el pare\n");
38                  sleep(1);
39            }
40
41            return (0);
42      }
```

05-varis.c

```
 1      #include <stdio.h>
 2      #include <pthread.h>
 3      #include <unistd.h>
 4
 5      /*Carles Pina i Estany  -  carles@pinux.info
 6       * Febrer 2004*/
 7
 8      /*Fem dos fills i s'executen paralelament*/
 9
10
11      void *codi (void *arg) {
12            int *i=(int*)arg;
13
14            pthread_detach(pthread_self());
15
16            for (;;) {
17                  printf("Sóc el fill %d\n",*i);
18                  sleep(1);
19            }
20
21            return (NULL);
22      }
23
24      int main () {
25            pthread_t thread[2];
26            int num1=0,num2=1;
27
28            pthread_create(&thread[0],NULL,&codi,&num1);
29            pthread_create(&thread[1],NULL,&codi,&num2);
```

```
30
31            sleep(5);
32            printf("Fi!\n");
33            return(0);
34
35      }
```

06-varis.c

```
 1      #include <stdio.h>
 2      #include <pthread.h>
 3      #include <unistd.h>
 4
 5      /*Carles Pina i Estany  -  carles@pinux.info
 6       * Febrer 2004*/
 7
 8      /*Dos threads paralels comparteixen una variable global*/
 9
10      int prova;
11
12      void *codi (void *arg) {
13            for (;;) {
14                  prova*=2;
15                  sleep(1);
16            }
17
18            return (NULL);
19      }
20
21
22      void *codi2 (void *arg) {
23            for (;;) {
24                  printf("Ara prova val: %d\n",prova);
25                  sleep(1);
26            }
27
28            return (NULL);
29      }
30
31      int main () {
32            pthread_t thread[2];
33
34            prova=2;
35
36            pthread_create(&thread[0],NULL,&codi,NULL);
37            pthread_create(&thread[1],NULL,&codi2,NULL);
38
39            sleep(5);
40            printf("Fi!\n");
41            return(0);
42
43      }
```

07-mutex.c

```
 1      #include <stdio.h>
 2      #include <pthread.h>
 3      #include <unistd.h>
 4      #include <string.h>
 5
 6      /*Carles Pina i Estany  -  carles@pinux.info
 7       * Febrer 2004*/
 8
 9      /*Exemple de mutex entre dos fils*/
10
11      typedef struct estructura_ {
12            pthread_mutex_t mutex;
13            int valor;
14      } estructura;
15
16      estructura data={PTHREAD_MUTEX_INITIALIZER,0};
17
18      int prova;
19      extern int errno;
20
21      void *codi (void *arg) {
22            int error=0;
23            sleep(1);
24
25            for (;;) {
26                  error=pthread_mutex_lock(&(data.mutex));
27                  //veure diferència entre _lock i _trylock
28                  printf("Error ha retornat: %d\n",error);
29                  printf("Error ha retornat: %s\n",strerror(error));
30
31
32                  printf("La dada val: %d\n",data.valor);
33                  pthread_mutex_unlock(&(data.mutex));
34            }
35
36            return (NULL);
37      }
38
39      void *codi2 (void *arg) {
40            for (;;) {
41                  printf("Ara el bloquejaré\n");
42                  pthread_mutex_lock(&(data.mutex));
43                  data.valor++;
44                  sleep(1);
45                  printf("Ara el desbloquejaré\n");
46                  pthread_mutex_unlock(&(data.mutex));
47                  sleep(1);
48            }
49
50            return (NULL);
51      }
52
53
54      int main () {
```

```
55            pthread_t thread;
56            prova=2;
57
58
59            /*data.mutex=PTHREAD_MUTEX_INITIALIZER;
60            data.valor=5;*/
61
62            pthread_create(&thread,NULL,&codi,NULL);
63            pthread_create(&thread,NULL,&codi2,NULL);
64
65            sleep(10);
66            printf("Fi!\n");
67            return(0);
68      }
```