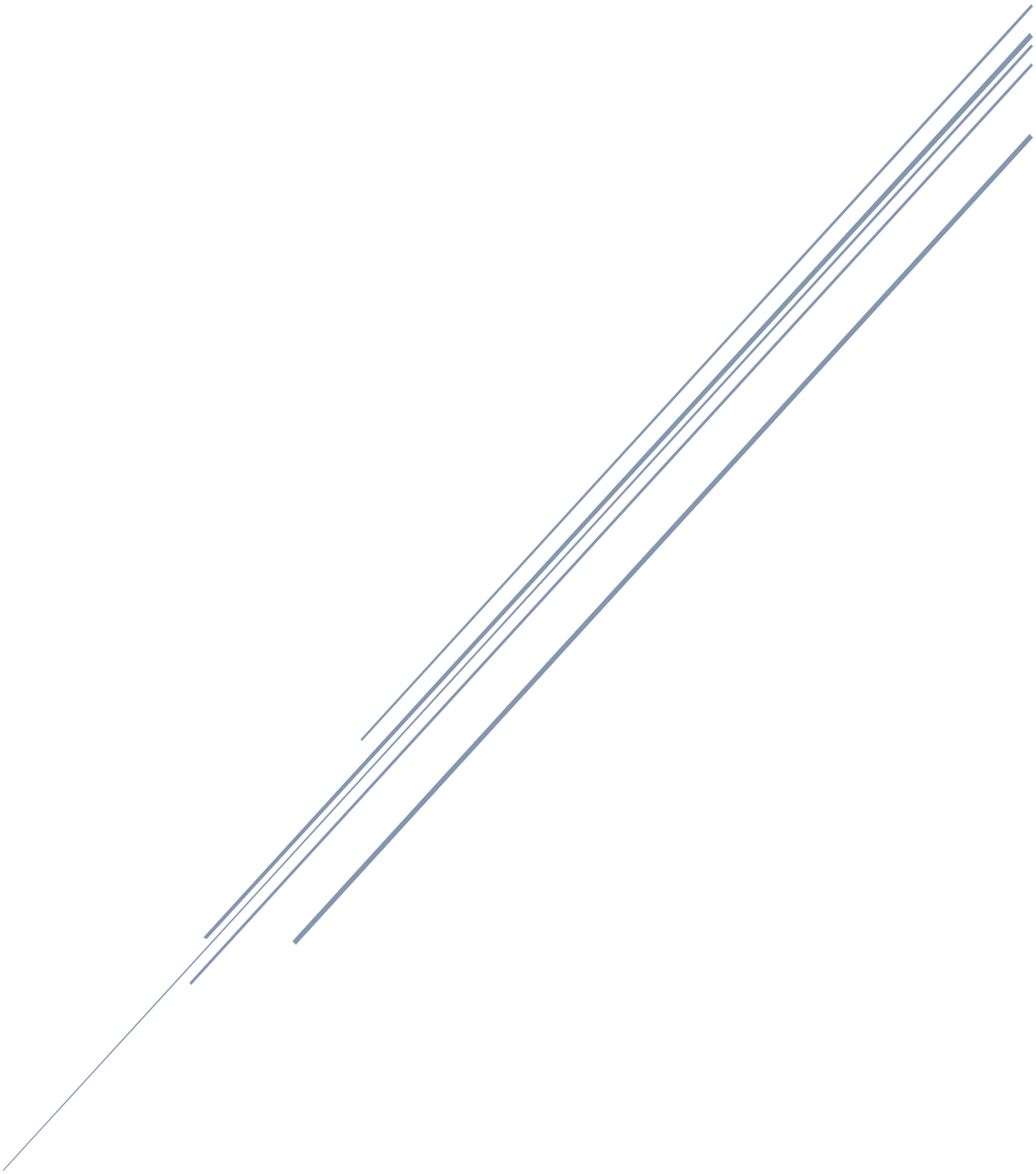
PRALAB1

Processos, pipes i senyals: Cronometratge campionats de natació



Escola Politécnica Superior UdL
Autor: Èric Bitriá Ribes

Índex

0. Introducció

En aquesta pràctica es realitza la implementació de sistemes de comunicació de processos basats en el sistema Linux, concretament en la versió 23.04 de Ubuntu, mitjançant la utilització de pipes i senyals entre processos pare i fill.

El programa a desenvolupar simula un campionat de natació, on el programa principal crearà un nombre de processos fills els quals actuaran de forma independent enviant i rebent dades amb el procés pare. Per tal de realitzar aquesta transferència de dades s'ha utilitzat processos de comunicació com les pipes i senyals accionades per l'usuari.

Consta de 4 versions on s'aniran introduint noves funcionalitats, en el transcurs d'aquest document s'aniran explicant els punts clau de les implementacions realitzades.

1. Versió 1

1.1 jutgePpal_v1

En aquesta primera implementació del programa es demanava executar tots els processos fills de forma concurrent amb el programa principal.

La versió inicial constava d'un bucle que anava creant cada procés fill, però tenia el problema que els creava de forma seqüencial, és a dir, creava un procés fill i esperava a la seva finalització abans de crear-ne un de nou. L'objectiu es crear el nombre de fills especificats de forma concurrent.

La part clau per realitzar-ho, era gracies a la comanda wait(): Aquesta comanda executada des del procés pare, espera a la finalització del procés fill. Simplement es requeria traure el wait() de dins del bucle que creava cada fill i realitzar els wait() necessaris en un bucle apart.

```
for (i = 0; i < NOMBRE_CARRILS; i++) {
    //...
    // Generar wait pels NOMBRE_CARRILS procesos
    child_pid = wait(&status);

    // Rebre els temps de cada proces a través del exitcode
    child_time = ((double)WEXITSTATUS(status)) / 10;
    sprintf(cadena, "Rebuda finalitzacio crono carril (pid-%u) ...");
    ImprimirInfoJutge(cadena);
     //...
}
```

Com podem veure en el codi realitzem un bucle for amb la comanda wait(), una vegada s'han rebut les dades mostrarà un missatge per pantalla i així fins completar el bucle.

1.2 cronoCarril_v1

En aquesta versió la implementació del cronoCarril no requeria de massa canvis, se'ns demanava retornar el temps de la simulació. La manera de fer-ho era a través del exitcode. D'aquesta manera la comanda wait del programa principal podria rebre el codi i transformar-ho en segons.

El problema d'utilitzar el exitCode es que solament accepta valors fins a 255, per tant tota simulació que tingués un temps superior a 25,5 segons mostraria un valor erroni.

2. Versió 2

2.1 jutgePpal_v2

En aquesta versió del codi es demana implementar les pipes per comunicar-se entre els diferents processos. En concret totes les dades que ens enviaran els carrils seran enviades mitjançant una pipe.

A més a més definirem una estructura de dades per facilitar l'enviament dels resultats, la qual contindrà totes les dades necessàries.

Per crear la pipe era necessari modificar el codi de creació de processos fills per a que aquests puguin heretar els descriptors de fitxers adequats:

```
// Creem array per la pipe

int pipeResultats[2];

// Inicialitzem la pipe

if(pipe(pipeResultats)==-1){

    //...
```

Una vegada inicialitzada la pipe, com que solament és el procés fill el que ha d'escriure les dades haurem de tancar o obrir els extrems de cada pipe depenent de si el procés és el pare o el fill:

```
// Tanquem la lectura del fill

close(pipeResultats[0]);

// Redireccionem sortida

dup2(pipeResultats[1],66);

// Tanquem la escritura del fill duplicada

close(pipeResultats[1]);
```

A més ens demanen que el procés fill escrigui per la sortida 66, simplement amb un dup2 realitzem aquesta especificació.

Ara solament hem de llegir la informació que envien els processos fill:

```
while(read(pipeResultats[0],&dadesResultatCrono,sizeof(t_resultat))>0)
{

//...
```

Amb la comanda read llegirem l'extrem de lectura de la pipe fins que aquesta no es tanqui, (el que significa que tots els carrils han enviat les seves dades).

2.2 cronoCarril_v2

El cronocarril emmagatzemarà les dades a enviar dins d'un structure i l'enviarà amb la comanda write a través de la sortida 66, seguidament farà un close(66).

```
if(write(66,&dadesResultatCrono,sizeof(t_resultat))<0){ //...
```

3. Versió 3

3.1 jutjePpal_v3

La versió 3 ens demana la implementació de codis únics per a cada procés fill i que aquest s'enviï a través d'una pipe individual per cada carril. Si hem de realitzar una pipe per cada carril crearem un array de pipes:

```
int pipeCodi[NOMBRE_CARRILS][2];
```

Dins de la creació dels processos fills executaríem la creació de cada pipe però ens trobem amb el problema que a cada execució estaríem duplicant per cada fill el nombre de pipes obertes. Per solucionar aquest problema la solució, que possiblement no es la més adient però igualment no se n' ha trobat una de millor, era tancar aquells descriptors de fitxers duplicats que no corresponguessin amb el procés fil:

```
// Tancar escriptura i lectura de les pipes que no necessitem
for (int j = 0; j < NOMBRE_CARRILS; j++) {
    if (j != i) {
       close(pipeCodi[j][0]);
       close(pipeCodi[j][1]);
```

A continuació tenim que generar un codi aleatori diferent per a cada carril, en la versió 3 es fa una implementació menys eficient que en la versió 4, però he decidit mantenir-la ja que en la versió 4 es fa un reestructuració major en tot el codi amb un únic array de dades per suportar les senyals, a més de mostrar el progrés al llarg de tota la pràctica.

```
for(int i = 0; i < NOMBRE_CARRILS; i++){
    dadesResultatCrono.codi = GenerarCodi(codis,i);
   if(write(pipeCodi[i][1],&dadesResultatCrono,sizeof(t_resultat))<0){
      //...
```

Aquesta part del codi genera i envia els codis a través de cada pipe al carril correponent.

```
int GenerarCodi(int codis[], int n){
    do {
        codi = rand() % 10000 + 250000;
        for (i = 0; i < n; i++) { // Mirar si ja existeix
            if (codi == codis[i]) {
                break;
//...
return codi;
```

I finalment la versió no final de generació de codis on primer comprovem la resta de codis generats per a que no es repeteixi.

3.2 cronoCarril_v3

El programa cronoCarril en aquest punt ha de llegir el codi generat pel codi principal i emmagatzemar-lo durant tot el transcurs de la simulació, per això abans de mostrar cap missatge ha de realitzar un read per rebre aquest codi:

```
while(read(99,&dadesPipe,sizeof(t_resultat))<0){

      ImprimirError("ERROR read pipe code");

};
```

El cronoCarril llegirà el codi enviat pel jutge i continuarà amb la seva execució normal.

4. Versió 4

4.1 jutgePpal_v4

La versió 4 suposa implementar l'execució del programa a través de senyals i tenint en compte que es poden executar múltiples sèries.

Per aconseguir aquesta funcionalitat s'ha requerit d'una reestructuració del programa incloent funcions especifiques per a cada part del programa, globalització de certes variables i la unió d'aquestes. Els elements més importants són aquests:

- **int GenerarCodi(int n):** La funció generar codis ja no requereix emmagatzemar i comprovar els codis dels altres carrils ja que reinicia la seva seed en base al pid de cada programa.
- **void EnviarCodis():** S'ha extret el codi d'enviar els codis a cada carril i reduït a una sola funció de tipus void.
- **void RebreDades():** El codi per rebre les dades i determinar el guanyador també s'ha situat com una funció apart.
- **void sigquit(int sig):** Aquesta funció realitzarà l'execució d'una nova sèrie.
- **void sigint(int sig):** Aquesta funció finalitzarà el programa, primer esperant a que s'acabin les series que puguin estar executant-se.

També s'han unificat les dades en una variable global en un array de structure per guardar pids, codis i numero de carril.

Pel que fa a la implementació de les senyals, el programa main es troba en un bucle infinit, ja que es la senyal sigint la que determinarà quan es finalitzarà el programa i la sigquit qui executarà les series:

```
void sigquit(int sig) {
    sprintf(cadena, "S'ha rebut SIGQUIT...”);
    //...
    // Enviem codis
    EnviarCodis();
    // Rebem dades i mostrem resultat del guanyador
    RebreDades();
}
```

Quan el programa rep la senyal sigquit (Ctrl+4 o \), mostrarà un missatge dient que ha rebut la senyal, seguidament enviarà els codis amb la funció EnviarCodis, i executarà la funció RebreDades que anirà mostrant els resultats de cada carril. Com que son execucions apart del programa principal aquesta senyal es pot executar les vegades que es vulguin permetent múltiples sèries.

```
void sigint(int sig) {
    sprintf(cadena, "S'ha rebut SIGINT ...");
    for(int i = 0; i < NOMBRE_CARRILS; i++){
        kill(dadesResultatCrono[i].pidCrono,SIGTERM);
    }
    if(dades == 1){
        // Rebre dades en cas que encara s'estigui executant
        RebreDades();
    }
    finalitzarCampionat();
```

La senyal sigint mostrarà el missatge de que s'ha rebut la senyal, i seguidament enviarà senyal SIGTERM a tots els procesos fill a través de la comanda kill. Com que tenim els pid emmagatzemats en les srtructure amb un bucle podem accedir fàcilment amb un for.

A continuació amb una variable global que s'actualitza per la funció RebreDades determinarem si encara esta executant-se una sèrie i si es el cas rebrem les seves dades. A continuació finalitzarem el campionat amb els wait corresponents i tancar els descriptors corresponents per notificar al cronoCarril.

4.2 cronoCarril_v4

El cronoCarril té la funció d'executar una sèrie si rep un codi, o de sortir en cas que el descriptor de lectura es tanqui. Amb la comanda read i tenint en compte que quan es tanca aquesta retorna el valor 0 podem aconseguir aquest comportament.

```
do{
    if((read_c = read(99, &codi, sizeof(int)))<0){
        ImprimirError("Read CODI");
    }else if(read_c == 0){
        endRace = 1;
}while(codi == 0 && endRace == 0);
if(endRace == 0){ //...
}else if(endRace == 1){
            break;
```

cronoCarril comprovarà quan el read retorni 0 i si es el cas la variable endRace agafarà el valor 1 i executarà un break per sortir del bucle de lectura del codi, finalitzant el programa i enviant el codi de finalització al programa principal.

També comentar que en rebre la senyal SIGTERM solament mostra un missatge, i les senyals SIGINT i SIGQUIT no realitzen res.