Práctica 4 (2.5 ptos)

PIPEs

1.− Objetivo

En esta práctica se va a aprender el uso de técnicas de comunicación entre procesos basadas en `pipe`s, así como métodos de atención a varios canales de comunicación distintos (`select`) y los efectos de dos rutinas de atención a peticiones diferentes.

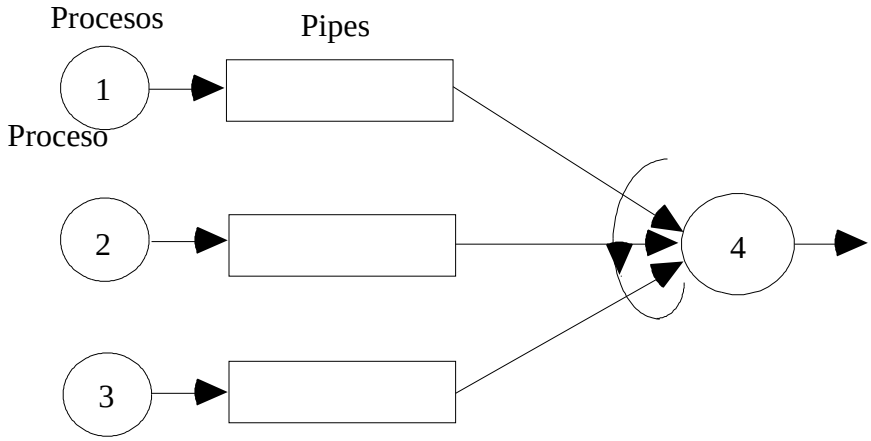
2.− `select()`

```
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds, struct timeval *timeout);
```

La función `select()` indica cuál de los descriptores de fichero especificados está listo para lectura, listo para escritura, o ha producido un error sin atender. Si esta condición es falsa para todos los descriptores de fichero especificados, `select()` se bloquea hasta que venza un timeout o se cumpla la condición para al menos uno de los descriptores de fichero especificados.

3.− Simulador

Se pretende crear un programa que cree la siguiente estructura de procesos:



La misión de los procesos 1−3 es la siguiente: Cada uno debe crear una estructura del tipo `pack`:

```
#define BYTES_SIN_USAR 100

typdef struct pack
{
   int  seq;
   struct timeval t;
   char unused_data[BYTES_SIN_USAR];
} pack;
```

Rellenarla y escribirla en su pipe correspondiente. El campo `seq` debe valer 1 para el primero que se cree e ir incrementando en una unidad con cada uno de los siguientes. El campo `t` es una marca de tiempo de cuándo se creó la estructura y `unused_dada` sirve tan solo para aumentar artificialmente el tamaño de la estructura. Cada proceso debe intentar escribir la estructura cuando le corresponde y caso de no poder no debe bloquearse haciéndolo sino que debe cejar en el intento (las pipes por defecto tienen un funcionamiento bloqueante, para obtener este otro modo de funcionamiento se puede activar su flag `O_NONBLOCK`).

El proceso 1 debe crear una estructura cada 3 segundos, el proceso 2 una cada 4 segundos y el proceso 3 una cada 5 segundos. Llamaremos instante de llegada de una estructura `pack` al instante en que es creada por su proceso correspondiente (el contenido de su campo `t`).

La tarea del proceso 4 va a ser extraer las estructuras `pack` enviadas por los procesos 1, 2 y 3 y procesarlas. Este proceso tendrá dos posibles modos de funcionamiento, que se diferenciarán en la forma de acceder a las pipes.

En el primer modo accederá secuencialmente a ellas, es decir, se bloqueará en la del proceso 1 a la espera de recibir un `pack`, cuando lo tenga lo procesará y pasará a la del proceso 2, esperará a leer un `pack` y lo procesará, después pasará a la del proceso 3, esperará a leer un `pack` y lo procesará, a continuación volverá a la pipe del proceso 1 y repetirá el procedimiento indefinidamente.

En el segundo modo de funcionamiento el proceso 4 se quedará bloqueado a la espera de que haya algo que leer por ALGUNA de las tres pipes. Cuando en alguna se hayan colocada datos los retirará, los procesará y repetirá el procedimiento.

Procesar un `pack`, en cualquiera de los dos modos, consistirá simplemente en suspenderse durante un segundo. Llamaremos instante de salida de una estructura `pack` al instante en que ha terminado de ser procesada por 4.

4.− Información obtenida por el proceso lector

Mediante el contenido de la estructura `pack` el proceso lector puede saber si se han perdido estructuras en la comunicación (gracias al campo `seq`) y cuánto tiempo ha transcurrido entre que una estructura se creó y fue procesada (gracias al campo `t`).

El proceso 4, para cada estructura `pack` que lea, debe sacar por pantalla una línea de texto conteniendo la siguiente información:

```
proc seq_llegada seq_leido t_llegada t_salida lost ret ret_medio
```

Donde el significado de los campos, separados por espacios o tabuladores, es el

siguiente:

`proc` un número del 1 al 3 indicando qué proceso creo la estructura.

`seq_llegada`: el contenido del campo `seq` de la estructura.

`seq_leido`: cuántas estructuras como ésta ha leido el proceso 4 hasta el momento de esa pipe.

`t_llegada`: contenido del campo `t` de la estructura (en segundos con decimales)

`t_salida`: instante en que terminó de ser procesada por 4 (en segundos con decimales).

`lost`: número de estructuras que, según los campos `seq`, se han perdido hasta el momento en esta pipe.

`ret`: retardo, tiempo transcurrido entre su llegada y su salida (en segundos con decimales).

`ret_medio`: el tiempo medio de retardo de las estructuras que han salido de esa pipe hasta el momento.

El programa simulador debe tener la siguiente sintaxis:

```
PIPS(1)                                                    PIPS(1)

NAME
    pips − Práctica 4

SYNOPSIS
    pips tipo

DESCRIPTION
    Si tipo vale 1 se empleará la primera técnica de atención a las pipes mencionada,
si vale 2 se empleará la segunda.
```

Ejemplo de salida por pantalla ejecutando `pips 1`

```
 1 1 1 938768113.582833 938768114.576349 0 0.993516
0.993516
 2 1 1 938768113.582983 938768115.576274 0 1.993291
1.993291
 3 1 1 938768113.584898 938768116.576315 0 2.991417
2.991417
 1 2 2 938768116.576312 938768117.576285 0 0.999973
0.996744
 2 2 2 938768117.576697 938768118.576290 0 0.999593
1.496442
 3 2 2 938768118.576327 938768119.576281 0 0.999954
1.995686
 1 3 3 938768119.576293 938768120.576303 0 1.000010
0.997833
 2 3 3 938768121.576290 938768122.576301 0 1.000011
1.330965
 3 3 3 938768123.576291 938768124.576311 0 1.000020
1.663797
 1 4 4 938768122.576558 938768125.576303 0 2.999745
1.498311
 2 4 4 938768125.576542 938768126.576313 0 0.999771
1.248166
 3 4 4 938768128.576478 938768129.576343 0 0.999865
```

```
1.497814
 1 5 5 938768125.576355 938768130.576339 0 4.999984
2.198646
 2 5 5 938768129.576730 938768131.576325 0 1.999595
1.398452
 3 5 5 938768133.576477 938768134.576358 0 0.999881
1.398227
                                  ...
```

Ejemplo de salida por pantalla ejecutando `pips 2`

```
 1 1 1 938768174.694560 938768175.696625 0 1.002065
1.002065
 2 1 1 938768174.694790 938768176.696552 0 2.001762
2.001762
 3 1 1 938768174.696682 938768177.696631 0 2.999949
2.999949
 1 2 2 938768177.686604 938768178.696577 0 1.009973
1.006019
 2 2 2 938768178.686660 938768179.696638 0 1.009978
1.505870
 3 2 2 938768179.696647 938768180.696586 0 0.999939
1.999944
 1 3 3 938768180.686602 938768181.696571 0 1.009969
1.007336
 2 3 3 938768182.686607 938768183.686598 0 0.999991
1.337244
 1 4 4 938768183.686858 938768184.686640 0 0.999782
1.005447
 3 3 3 938768184.696586 938768185.696608 0 1.000022
1.666637
 1 5 5 938768186.686621 938768187.686623 0 1.000002
1.004358
 2 4 4 938768186.686641 938768188.686614 0 1.999973
1.502926
 1 6 6 938768189.686606 938768190.686635 0 1.000029
1.003637
 3 4 4 938768189.696599 938768191.686624 0 1.990025
1.747484
 2 5 5 938768190.686944 938768192.686645 0 1.999701
1.602281
                                    ...
```

5.− Ficheros

En el directorio `$(HOME)/solucion/prac4` debe encontrarse un `Makefile` así como todos los ficheros `.c` y `.h` necesarios para crear `pips`. La acción por defecto del `Makefile` (la cual debe funcionar con solo hacer `make` en ese directorio) debe ser crear este programa. Igualmente debe haber los siguientes ficheros en el directorio en formato gráfico PPM: `perdidos_1.ppm`, `perdidos_2.ppm`, `retardo_medio_1.ppm`, `retardo_medio_2.ppm`.

El contenido de estos gráficos es el siguiente:

`perdidos_1.ppm`: incluye tres gráficas, una para cada pipe, mostrando el número de paquetes perdidos con el primer método respecto al número de paquete de salida de esa pipe.

`perdidos_2.ppm`: incluye tres gráficas, una para cada pipe, mostrando el número de paquetes perdidos con el segundo método respecto al número de paquete de salida de esa pipe.

`retardo_medio_1.ppm`: incluye tres gráficas, una para cada pipe, mostrando el retardo medio de los paquetes de esa pipe con el primer método respecto al número de paquete de salida de esa pipe.

r`etardo_medio_2.ppm`: incluye tres gráficas, una para cada pipe, mostrando el

retardo medio de los paquetes de esa pipe con el segundo método respecto al número de paquete de salida de esa pipe.

En todos los casos la simulación realizada debe durar el tiempo suficiente para que el funcionamiento del sistema se estabilice o quede claro su carácter inestable.

Para la corrección de la práctica se borrarán todos los ejecutables y se recompilará mediante el `Makefile`.

6.− Aplicaciones, funciones y llamadas al sistema útiles

`gnuplot(1), awk(1), cut(1), pipe(2), fcntl(2), select(2), sleep(3), gettimeofday(2).`