Info_sw
www.wuolah.com/student/Info_sw

772

Resumen_APSO_programacion.pdf

Resumen_APSO_programacion

2º Administración y Programación de Sistemas Operativos

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería
UHU - Universidad de Huelva

APSO. PROGRAMACIÓN. RESUMEN. CURSO 17/18.

CREACIÓN DE PROCESOS.

NO OLVIDARSE AÑADIR LAS LIBRERÍAS NECESARIAS

```
main()
{
    int pid_hijo;
    ...
    pid_hijo = fork();
    if (pid_hijo==0) // Soy la copia, hago execl para crear al hijo.
    {
        execl("proceso_hijo","proceso_hijo",NULL); // "proceso_hijo" es el nombre del ejecutable generado
                                                   // a partir de fichero "proceso_hijo.c".
        perror("Error de execl\n");
        exit(-1);
    }
    else if(pid_hijo==-1)
        printf("Imposible hacer el fork\n");
    else // Soy el proceso original, hago lo que sea...
    {
        ... Opciones: (entre otras)
    }
}
```

Opciones:
- Crear otros procesos
- Enviar/esperar señales
- Envío mensajes en colas de mensajes (o recibir los mismos)
- Escribir/Leer en FIFOS
- Leer/escribir en tuberías (pipes)

SINCRONIZACIÓN PADRE-HIJO.

`wait()` → llamada que ejecuta el proceso padre (original).
`wait(NULL);` → Espera a que termine uno de sus hijos.

`exit()` o `exit(NULL);` → Llamada que ejecuta siempre un proceso hijo para terminar.

Preparar a un proceso para recibir una señal:

Dos opciones:

1ª OPCIÓN:
```
struct sigaction s1;
s1.sa_flags = 0;
s1.sa_handler = rutina10;
sigaction(10,&s1,NULL);
```

2ª OPCIÓN:
```
signal(12, rutina12);
```

Enviar una señal:

```
kill(pid_proceso, 10);
```

`alarm(x);` → el proceso que haga un alarm(x) recibirá la señal 14 cada x segundos. Por tanto, este proceso debe estar preparado para recibir la señal 14, creando además la rutina para tratar la interrupción de la señal 14.

`alarm(0);` → anula el alarm.

`pause();` → espera un proceso hasta que llegue una señal.

`sleep(x);` → espera parado una cantidad x de segundos. Si durante un "sleep" llega una señal, los segundos restantes del sleep se los "salta".

FIFOS.

Creación de la FIFO. → Lo hará un único proceso.

```
mkfifo ("fifo1", 0777);   // Nombre de la fifo + permisos (con un 0 delante).
```

Borrar la FIFO: → Lo hará un único proceso (normalmente el que termine el último).

```
unlink ("fifo1");   // Nombre de la fifo entre comillas.
```

Todos los procesos que lean/escriban en la fifo deben hacer:

```
int fd;
fd = open ("fifo1", O_RDWR);
```

Para leer, tras hacer las dos líneas anteriores:

```
read (fd, &variable, sizeof(variable));
```

Para escribir, tras hacer las dos líneas anteriores:

```
write (fd, &variable, sizeof(variable));
```

```
close(fd);
```
No sé si habría que hacerlo tras leer/escribir en la FIFO. Si lo hago da error y si no lo hago funciona.

TUBERÍAS (pipes).

Creación de una tubería:

```
int tubo[2];
pipe (tubo);
```

→ en `tubo[0]` guarda el puntero de lectura de la tubería y en `tubo[1]` guarda el puntero de escritura en la tubería.

(3) posición en la que guarda el P.L. en la tabla de canales)

(4) posición en la que guarda el P.E. en la tabla de canales).

Para poder leer/escribir en la tabla de canales, el proceso debe tener acceso a los P.L. y/o P.E. de la tubería (es decir, a tub[0] o tub[1]). Si el proceso que lee o escribe es un proceso hijo creado por "`execl(_,_,NULL)`", entonces tendremos que colocar el P.L. o P.E en la entrada 2 de la tabla de canales de la siguiente manera:

```
close(2);
dup(tubo[0]);   ó   dup(tubo[1]);
```

Para duplicar puntero de lectura

para duplicar puntero de escritura.

Leer: `read(tubo[0], &variable, sizeof(variable));` (*1)
`read(2, &variable, sizeof(variable));` (*2)

Escribir: `write(tubo[1], &variable, sizeof(variable));` (*1)
`write(2, &variable, sizeof(variable));` (*2)

(*1) Así lee/escribe el proceso original.

(*2) Así lee/escribe un proceso hijo creado con execl que hereda sólo las 3 primeras entradas de la tabla de canales.

RESUMEN COLAS DE MENSAJES.

man ipc
Para recordar los comandos de las colas (no están todos).

Todos los procesos que trabajen con la cola deben hacer esto:

```
struct Mensaje1 m1;
struct Mensaje2 m2;
```
} Definir los struct necesarios

```
key_t clave;
int id_cola;
clave = ftok("./fichero", 18);
if (clave == (key_t)-1) { perror(...); exit(-1); }
id_cola = msgget(clave, 0600 | IPC_CREAT);
if (id_cola == -1) { perror(...); exit(-1); }
```

→ Fichero creado para hacer ftok.
→ Nº cualquiera
Los mismos en todos los procesos.

Fuera del main():

```
struct Mensaje1
{ long tipo;
  char palabra[10];
  int numero;
};
struct Mensaje2
{ long tipo;
  char letra;
};
```

> Para leer o recibir mensajes:

```
msgrcv(id_cola, (struct msgbuf *)&m1, sizeof(m1)-sizeof(long), 1, 0);
```
1 → tipo de mensaje

> Para escribir o enviar mensajes:

```
m2.tipo = 2;
m2.letra = a;
```
} Rellenamos los campos del struct.

```
msgsnd(id_cola, (struct msgbuf *)&m2, sizeof(m2)-sizeof(long), IPC_NOWAIT);
```

Por último, uno de los procesos que intervienen en la lectura/escritura en la cola de mensajes debe eliminarla:

```
msgctl(id_cola, IPC_RMID, (struct msqid_ds *)NULL);
```

VER SI HEMOS DEJADO COLAS DE MENSAJES CREADAS EN EL SISTEMA:

```
> ipcs -q
```

ELIMINAR UNA COLA DEL SISTEMA:

```
> ipcrm msg identificador_cola
```