Procesos - IPCs

Ing Florencia Ferrigno

2016

UTN - FRBA

Un poco de repaso

- Un proceso es una instancia de ejecución de un programa
- El task scheduler distribuye los tiempos de ejecucion entre todos los procesos que estan en la tabla de procesos.
- Los procesos tienen atributos o caracteristicas que los identifican
- Los atributos basicos son su identificador o PID (process ID) y el identificador del proceso padre del mismo PPID (parent process ID)

Parentesco entre procesos

- Cuando un proceso crea uno nuevo, decimos que se creo un proceso hijo, y por lo tanto su creador sera el proceso padre.
- Se puede trazar una ascendencia de todos los procesos hasta llegar al proceso que tiene **PID = 1**: Este proceso se lo conoce como **init**.
- **init** es el primer proceso que tiene lugar luego que arranca el kernel
- **init** es quien pone en funcionamiento el sistema, levanta todos los otros procesos.

Creacion de procesos

- La funcion **pid_t fork()** permite crear un nuevo proceso

Creacion de procesos

- La funcion **pid_t fork()** permite crear un nuevo proceso
- Quien llame a esta funcion se lo conoce como proceso **padre** y el proceso creado se lo conoce como **hijo**

Creacion de procesos

- La funcion **pid_t fork()** permite crear un nuevo proceso
- Quien llame a esta funcion se lo conoce como proceso **padre** y el proceso creado se lo conoce como **hijo**
- El **hijo** obtiene una copia de los datos del proceso padre.

Creacion de procesos

- La funcion **pid_t fork()** permite crear un nuevo proceso
- Quien llame a esta funcion se lo conoce como proceso **padre** y el proceso creado se lo conoce como **hijo**
- El **hijo** obtiene una copia de los datos del proceso padre.
- El valor de retorno de esta funcion puede ser:
  - **0**: este valor de retorno indica que es el proceso hijo. En caso de querer saber cual es el PID del mismo basta con llamar a la fucnion: **getpid()** y en caso de necesitar el PID del proceso padre basta con llamar a la funcion: **getppid()**
  - **-1**: indica que algo salio mal y proceso hijo no fue creado. Podemos usar **perror()** para saber que paso. Probablemente sea por que la tabla de procesos esta llena y no hay mas lugar para crear un proceso nuevo.
  - **cualquier otro valor mayor a cero**: este valor de retorno indica que es el proceso padre y el valor retornado es

Creacion de procesos

- No se puede precedecir si un proceso padre se ejecutara antes o despues que el proceso hijo.

Creacion de procesos

- No se puede precedecir si un proceso padre se ejecutara antes o despues que el proceso hijo.
- El programa se ejecuta fuera de secuencia, es decir asincronicamente.

Creacion de procesos

- No se puede precedecir si un proceso padre se ejecutara antes o despues que el proceso hijo.
- El programa se ejecuta fuera de secuencia, es decir asincronicamente.
- La naturaleza asincronica del fork significa que no se debera ejecutar codigo perteneciente al proceso hijo que dependa de la ejecucion del codigo del proceso padre.

Creacion de procesos

- No se puede precedecir si un proceso padre se ejecutara antes o despues que el proceso hijo.
- El programa se ejecuta fuera de secuencia, es decir asincronicamente.
- La naturaleza asincronica del fork significa que no se debera ejecutar codigo perteneciente al proceso hijo que dependa de la ejecucion del codigo del proceso padre.

Espera de procesos

- Cuando un proceso muere (termina su ejecucion) no figura mas en la lista de procesos. Sin embargo, una pequeña parte de el permanece a la espera que el padre se haga cargo.

Espera de procesos

- Cuando un proceso muere (termina su ejecucion) no figura mas en la lista de procesos. Sin embargo, una pequeña parte de el permanece a la espera que el padre se haga cargo.
- Luego de generar un nuevo proceso, el proceso padre debe esperar para que su proceso hijo termine a fin de recoger su condicion de salida y evitar la creacion de procesos zombies

Espera de procesos

- Cuando un proceso muere (termina su ejecucion) no figura mas en la lista de procesos. Sin embargo, una pequeña parte de el permanece a la espera que el padre se haga cargo.
- Luego de generar un nuevo proceso, el proceso padre debe esperar para que su proceso hijo termine a fin de recoger su condicion de salida y evitar la creacion de procesos zombies
- Para lograr la espera contamos con dos funciones: **waitpid()** y **wait()**

Espera de procesos

- Cuando un proceso muere (termina su ejecucion) no figura mas en la lista de procesos. Sin embargo, una pequeña parte de el permanece a la espera que el padre se haga cargo.
- Luego de generar un nuevo proceso, el proceso padre debe esperar para que su proceso hijo termine a fin de recoger su condicion de salida y evitar la creacion de procesos zombies
- Para lograr la espera contamos con dos funciones: **waitpid()** y **wait()**
- El padre puede tener multiples hijos, entonces la pregunta es: como sabe a cual de todos ellos debe esperar? eso lo logramos a traves de **waitpid()**

Qué es un proceso zombie?

- Cuando un proceso hijo termina sin que su proceso padre disponga recoger la condicion del mismo

Qué es un proceso zombie?

- Cuando un proceso hijo termina sin que su proceso padre disponga recoger la condicion del mismo
- Un proceso padre recoge la condicion de salida de un proceso hijo usando una de las dos funciones de wait.

Qué es un proceso zombie?

- Cuando un proceso hijo termina sin que su proceso padre disponga recoger la condicion del mismo
- Un proceso padre recoge la condicion de salida de un proceso hijo usando una de las dos funciones de wait.
- Se lo denomina **zombie** por que efectivamente esta muerto, si ejecutamos el comando ps figura en la tabla de procesos pero con una Z

Qué es un proceso zombie?

- Cuando un proceso hijo termina sin que su proceso padre disponga recoger la condicion del mismo
- Un proceso padre recoge la condicion de salida de un proceso hijo usando una de las dos funciones de wait.
- Se lo denomina **zombie** por que efectivamente esta muerto, si ejecutamos el comando ps figura en la tabla de procesos pero con una Z
- El proceso hijo ha terminado, se libero memoria y demas recursos asignados, pero aun ocupa un lugar en la tabla de procesos del kernel, quien almacena la condicion de salida del proceso hijo hasta que el proceso padre lo retire de ahi

Que es un proceso huerfano?

- Es un proceso hijo cuyo padre termina antes de llamar a **wait()** o **waitpid()**

Que es un proceso huerfano?

- Es un proceso hijo cuyo padre termina antes de llamar a **wait()** o **waitpid()**
- En este caso el proceso init asume el papel de padre del proceso huerfano y recoge su condicion de salida, evitando en consecuencia la aparicion del proceso zombie.

wait() y waitpid()

- Headers:
  - **sys/types.h**
  - **sys/wait.h**
- Prototipos:
  - **pid_t wait(int* status)**
  - **pid_t waitpid(pid_t pid, int* status, int options)**
- **NOTA**: Estas funciones son usadas para esperar el cambio de estado de un proceso hijo. Si el mismo ya cambio de estado, entonces estas funciones devolveran el control al proceso padre de manera inmediata. En caso contrario, el proceso padre queda bloqueado hasta que el proceso hijo cambie de estado (ya sea que termino o que fue abortado mediante alguna señal).
- **wait(&status)** es equivalente a hacer **waitpid(-1,&status,0)**
- Donde la variable status es quien tendra almacenado el estado de terminacion del proceso hijo.

Que son las señales?

- Son analogas a las interrupciones, es decir no sabemos cuando un proceso recibira alguna señal

Que son las señales?

- Son analogas a las interrupciones, es decir no sabemos cuando un proceso recibira alguna señal
- Es un suceso que tiene lugar en cualquier momento durante la ejecucion de un proceso.

Que son las señales?

- Son analogas a las interrupciones, es decir no sabemos cuando un proceso recibira alguna señal
- Es un suceso que tiene lugar en cualquier momento durante la ejecucion de un proceso.
- Son asincronicas, es decir que cuando y como llegaran es algo impredecible.

Que son las señales?

- Son analogas a las interrupciones, es decir no sabemos cuando un proceso recibira alguna señal
- Es un suceso que tiene lugar en cualquier momento durante la ejecucion de un proceso.
- Son asincronicas, es decir que cuando y como llegaran es algo impredecible.
- Puede ser usadas como una via de comunicacion entre procesos.

Que son las señales?

- Son analogas a las interrupciones, es decir no sabemos cuando un proceso recibira alguna señal
- Es un suceso que tiene lugar en cualquier momento durante la ejecucion de un proceso.
- Son asincronicas, es decir que cuando y como llegaran es algo impredecible.
- Puede ser usadas como una via de comunicacion entre procesos.
- Hasta ahora los procesos no tienen ningun control sobre ellas.

Señales

- Es la via mas simple de comunicacion entre procesos.

Señales

- Es la via mas simple de comunicacion entre procesos.
- No se pueden transmitir datos, sino que a traves de un servicio del kernel, un proceso da aviso al otro de la ocurrencia de algun evento.

Señales

- Es la via mas simple de comunicacion entre procesos.
- No se pueden transmitir datos, sino que a traves de un servicio del kernel, un proceso da aviso al otro de la ocurrencia de algun evento.
- Como respuesta a dicha señal, se tendra una accion predeterminada

Señales

- Es la via mas simple de comunicacion entre procesos.
- No se pueden transmitir datos, sino que a traves de un servicio del kernel, un proceso da aviso al otro de la ocurrencia de algun evento.
- Como respuesta a dicha señal, se tendra una accion predeterminada
- Existen 32 señales que se pueden enviar de un proceso a otro, y cada una de ellas tiene un comportamiento determinado.Si queremos conocer la lista podemos ejecutar el comando **kill -l** desde el shell

Señales

- Es la via mas simple de comunicacion entre procesos.
- No se pueden transmitir datos, sino que a traves de un servicio del kernel, un proceso da aviso al otro de la ocurrencia de algun evento.
- Como respuesta a dicha señal, se tendra una accion predeterminada
- Existen 32 señales que se pueden enviar de un proceso a otro, y cada una de ellas tiene un comportamiento determinado.Si queremos conocer la lista podemos ejecutar el comando **kill -l** desde el shell
- El kernel nos provee servicios para que el proceso receptor pueda modificar dicho comportamiento.

Señales

- Cada señal tiene un nombre que la identifica, todas ellas comienzan con **SIG**

Señales

- Cada señal tiene un nombre que la identifica, todas ellas comienzan con **SIG**
- Dichos nombres corresponden a valores enteros conocidos como numero de señal, las mismas estan definidas en **signal.h**

Señales

- Cada señal tiene un nombre que la identifica, todas ellas comienzan con **SIG**
- Dichos nombres corresponden a valores enteros conocidos como numero de señal, las mismas estan definidas en **signal.h**
- Cuando un proceso recibe la señal, tiene 3 opciones:
  - Ignorarla
  - Dejar que ocurra la accion predeterminada asociada a la señal
  - Capturar la señal (mediante un handler) y manipular la misma segun los intereses del proceso en cuestion.

Señales

- **SIGSTOP**: enviar esta señal hace que el proceso que la recibe pare con sus tareas.

Señales

- **SIGSTOP**: enviar esta señal hace que el proceso que la recibe pare con sus tareas.
- **SIGCONT**: enviar esta señal a un proceso que ha parado sus tareas hace que reanude su trabajo.

Señales

- **SIGSTOP**: enviar esta señal hace que el proceso que la recibe pare con sus tareas.
- **SIGCONT**: enviar esta señal a un proceso que ha parado sus tareas hace que reanude su trabajo.
- **SIGINT**: Esta señal es la que se genera cuando hacemos **Ctrl** + **C** y provoca el fin del proceso. Podemos sobreescribir dicha señal para que nuestro proceso haga lo que querramos al recibir esta señal.

UTN.BA

Señales

- **SIGSTOP**: enviar esta señal hace que el proceso que la recibe pare con sus tareas.
- **SIGCONT**: enviar esta señal a un proceso que ha parado sus tareas hace que reanude su trabajo.
- **SIGINT**: Esta señal es la que se genera cuando hacemos **Ctrl + C** y provoca el fin del proceso. Podemos sobreescribir dicha señal para que nuestro proceso haga lo que querramos al recibir esta señal.
- **SIGKILL**: Cada vez que se tipea **kill -9 pid** lo que estamos haciendo es enviar la señal SIGKILL al proceso cuyo pid es que se se paso. **Esta señal no puede ser interceptada ni ignorada, al igual que SIGSTOP**

Señales

- **SIGSTOP**: enviar esta señal hace que el proceso que la recibe pare con sus tareas.
- **SIGCONT**: enviar esta señal a un proceso que ha parado sus tareas hace que reanude su trabajo.
- **SIGINT**: Esta señal es la que se genera cuando hacemos **Ctrl + C** y provoca el fin del proceso. Podemos sobreescribir dicha señal para que nuestro proceso haga lo que querramos al recibir esta señal.
- **SIGKILL**: Cada vez que se tipea **kill -9 pid** lo que estamos haciendo es enviar la señal SIGKILL al proceso cuyo pid es que se se paso. **Esta señal no puede ser interceptada ni ignorada, al igual que SIGSTOP**
- **SIGTERM**: Cuando ejecutamos kill sin la opcion -9 la señal que estamos enviando es SIGTERM. Esta puede tener sobreescrito su handler de modo que podemos hacer una terminacion mas prolija de nuestro proceso.

UTN.BA

Señales

- **SIGSTOP**: enviar esta señal hace que el proceso que la recibe pare con sus tareas.
- **SIGCONT**: enviar esta señal a un proceso que ha parado sus tareas hace que reanude su trabajo.
- **SIGINT**: Esta señal es la que se genera cuando hacemos **Ctrl + C** y provoca el fin del proceso. Podemos sobreescribir dicha señal para que nuestro proceso haga lo que querramos al recibir esta señal.
- **SIGKILL**: Cada vez que se tipea **kill -9 pid** lo que estamos haciendo es enviar la señal SIGKILL al proceso cuyo pid es que se se paso. **Esta señal no puede ser interceptada ni ignorada, al igual que SIGSTOP**
- **SIGTERM**: Cuando ejecutamos kill sin la opcion -9 la señal que estamos enviando es SIGTERM. Esta puede tener sobreescrito su handler de modo que podemos hacer una terminacion mas prolija de nuestro proceso.

Señales

- **SIGUSR1** y **SIGUSR2**: Son dos señales que no tienen definida ninguna accion, estan disponibles para que el programador las use a su antojo.

Señales

- **SIGUSR1** y **SIGUSR2**: Son dos señales que no tienen definida ninguna accion, estan disponibles para que el programador las use a su antojo.
- **SIGALRM**: timer generada por la funcion **alarm** del sistema. La funcion alarm pone a correr el timer con un intervalo de tiempo con la que fue programada. Cuando dicho tiempo se vence se levanta esta señal. La accion predeterminada es la finalizacion del proceso.

Señales

- **SIGUSR1** y **SIGUSR2**: Son dos señales que no tienen definida ninguna accion, estan disponibles para que el programador las use a su antojo.
- **SIGALRM**: timer generada por la funcion **alarm** del sistema. La funcion alarm pone a correr el timer con un intervalo de tiempo con la que fue programada. Cuando dicho tiempo se vence se levanta esta señal. La accion predeterminada es la finalizacion del proceso.
  - Como setear la alarma: **unsigned int alarm (unsigned int seconds)**. El valor de retorno sera 0 si no se programo ninguna otra alarma o el numero de segundos remanentes de una alarma previamente programada. **Un proceso solo puede tener una UNICA alarma programada**

Señales

- **SIGUSR1** y **SIGUSR2**: Son dos señales que no tienen definida ninguna accion, estan disponibles para que el programador las use a su antojo.
- **SIGALRM**: timer generada por la funcion **alarm** del sistema. La funcion alarm pone a correr el timer con un intervalo de tiempo con la que fue programada. Cuando dicho tiempo se vence se levanta esta señal. La accion predeterminada es la finalizacion del proceso.
  - Como setear la alarma: **unsigned int alarm (unsigned int seconds)**. El valor de retorno sera 0 si no se programo ninguna otra alarma o el numero de segundos remanentes de una alarma previamente programada. **Un proceso solo puede tener una UNICA alarma programada**

Captura de eventos de señales

- **void (*signal(int sig, void (*func)(int))) (int)**

Captura de eventos de señales

- **void (*signal(int sig, void (*func)(int))) (int)**
- Pasa la señal para que sea atendida por la funcion apuntada por el puntero a funcion (func). Dicho puntero a funcion es una funcion que toma como argumento un entero (el numero que representa la señal que atiende) y devuelve void.

Como enviar señales

- **int kill(pid_t pid, int sig)**:
- Headers:
  - **sys/types.h**
  - **signal.h**
- Es un system call que puede ser usada para enviar cualquier señal a cualquier proceso.
- Valores del pid:
  - Si el valor es positivo la señal sera enviada al proceso que tenga dicho PID.
  - Si es 0, la señal es enviada a todo proceso que pertenece al mismo grupo que el proceso llamante
  - Si es -1, la señal es enviada a todo proceso al cual el llamante tiene permiso de enviar la señal, salvo el proceso init.
  - Si es cualquier valor menor a -1, la señal se envia a todo proceso perteneciente al grupo cuyo pid sea -pid
- Si sig es cero, ninguna señal es enviada. Se puede usar para chequear la existencia del proceso cuyo pid se envio como primer argumento.
- Valor de retorno:

UTN.BA

Pipes

- Las señales si bien son un buen metodo de comunicarse con otro proceso, no nos permiten el intercambio de datos entre ellos.
- Para poder intercambiar informacion entre procesos, lo mas sencillo son los **pipes**
- Un **pipe** es un nodo en el file system que se comporta como un archivo.
- Son dispositivos FIFO (primero en entrar, primero en salir), los datos escritos en el pipe se leen de manera secuencial, comenzando con el primer byte y se termina con el ultimo byte escrito.
- Los datos leidos son extraidos del pipe, es lectura destructiva.

Pipes

- Los pipes se crean mediante la system call: **int pipe(int fd[2])**
- Crea un pipe, un canal unidireccional, utilizado para la comunicacion entre procesos.
- Header: **unistd.h**
- El vector, es un vector de dos descriptores:
  - fd[0]: descriptor de lectura.
  - fd[1]: descriptor de escritura.
- Valores de retorno:
  - 0: valor de retorno en caso de exito.
  - -1: en caso de error.

Limitaciones

- Solo sirve para comunicarse entre procesos padre e hijo.
- Solo se puede crear el pipe, dentro de la instancia del proceso padre antes de crear el proceso hijo.
- Es una comunicacion half duplex, es decir que existe solo un sentido de comunicacion.
- Se recomienda que cada proceso cierre el descriptor que no va a usar, es decir que si el proceso padre es quien lee, debera cerrar el descriptor de escritura y viceversa.

Pipes

- Operador |: Conecta la salida de un comando con la entrada del siguiente. En otras palabras, aplica una redireccion del stdout del primer proceso, al stdin del siguiente proceso.
- Si probamos hacer desde el shell: **ls | wc -l**, lo que hara sera que la salida del ls en lugar de aparecer en pantalla sera la entrada para el proceso wc que con la opcion -l contara la cantidad de lieas.
- Vamos a intentar a hacer este programa. Para lo cual vamos a necesitar usar las siguientes funciones:

Pipes

- **exec()**: reemplaza completamente el proceso que efectuo la llamada.Su PID no varia. Sus argumentos son:
  - **path**: ruta completa de acceso al archivo binario ejecutable o al script que se desea ejecutar.
  - **argv**: es la lista completa de argumentos que se le desea transferir al programa, incluyendo argv[0], que ha sido tradicionalmente el nombre del programa a ser ejecutado.
  - **envp**: puntero a un entorno especializado, si lo hubiera, para el programa que debe iniciar exec.
- La familia exec consta de 6 funciones:
  - 4 de ellas son: execl, execv, execle, execve.
  - las dos restantes son: execlp y execvp. Estas son las que mas usamos.
- Las funciones que contienen **l**:esperan una lista de argumentos separados por coma, terminada por un puntero a NULL.
- Las funciones que contienen **v**:admiten un vector, un array de punteros a strings, este array debe ser terminado con un puntero a NULL.

UTN.BA

Exec

- Supongamos que queremos ejecutar el comando **cat** (concatena archivos y los muestra por la salida standard), con los siguientes argumentos:
  - /etc/passwd
  - /etc/group
- Usando una de las funciones con **l**, simplemente habria que transferir cada uno de estos valores:
  - **execl("/bin/cat","/etc/passwd","/etc/group",NULL)**
- Usando una de las funciones con **v**, primero debemos crear el array argv y luego pasarlo a la funcion.
  - **char *argv[] = "/bin/cat","/etc/passwd","/etc/group",NULL**
  - **execv("/bin/cat",argv)**

dup()

- Header: **unistd.h**
- **int dup(int oldfd)**
- Es un system call que recibe un descriptor de archivos abierto y lo duplica.