Introducción a los Sistemas Operativos

Procesos - IV

Profesores:
Lía Molinari
Juan Pablo Pérez
Macia Nicolás

Facultad de Informática
UNIVERSIDAD NACIONAL DE LA PLATA

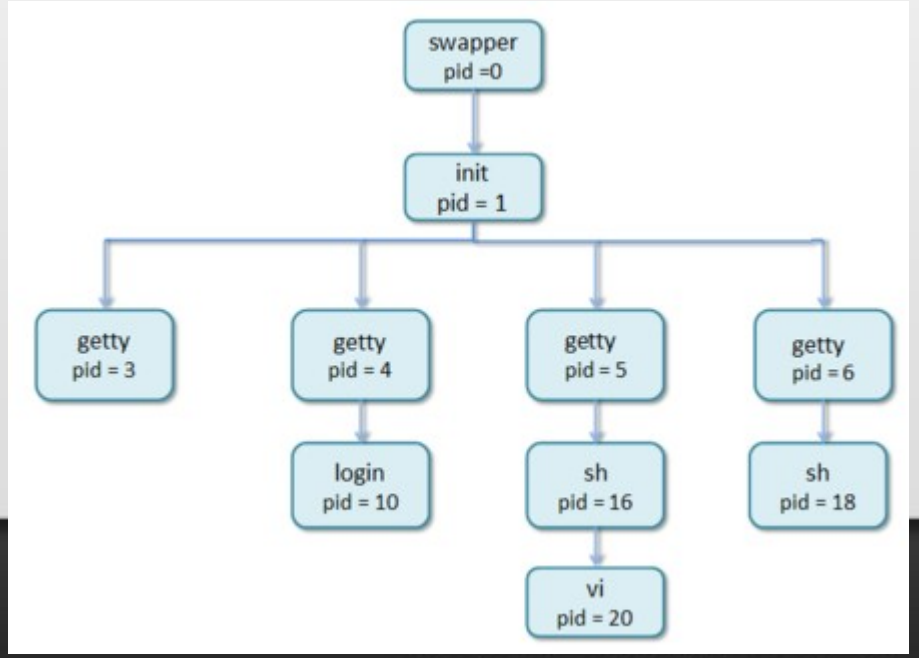
I.S.O.

- ☑ Versión: Agosto 2015
- ☑ Palabras Claves: Procesos, Linux, Windows, Creación, Terminación, Fork, Execve, Relación entre procesos

Los temas vistos en estas diapositivas han sido mayormente extraídos del libro de Andrew S. Tanenbaum (Sistemas Operativos Modernos)

Creación de procesos

- Un proceso es creado por otro proceso
- Un proceso padre tiene uno o más procesos hijos.
- Se forma un árbol de procesos



Actividades en la creación

- ☑ Crear la PCB
- ☑ Asignar PID (Process IDentification) único
- ☑ Asignarle memoria para regiones
  - Stack, Text y Datos
- ☑ Crear estructuras de datos asociadas
  - Fork (copiar el contexto, regiones de datos, text y stack)

Relación entre procesos Padre e Hijo

Con respecto a la Ejecución:

- ☑ El padre puede continuar ejecutándose concurrentemente con su hijo
- ☑ El padre puede esperar a que el proceso hijo (o los procesos hijos) terminen para continuar la ejecución.

Relación entre procesos Padre e Hijo (cont.)

Con respecto al Espacio de Direcciones:

- ☑ El hijo es un duplicado del proceso padre (caso Unix)
- ☑ Se crea el proceso y se le carga adentro el programa (caso Windows)

Creación de Procesos

- En UNIX:
  - system call **fork()** crea nuevo proceso
  - system call **execve(),** usada después del fork, carga un nuevo programa en el espacio de direcciones.
- En Windows:
  - system call **CreateProcess()** crea un nuevo proceso y carga el programa para ejecución.

Uso de la system call fork

```
#
# El padre puede terminar antes que los hijos
#

import os, time
hijos = 0
print '\n\n\nSoy el PROCESO PADRE. PID: ', os.getpid() , 'y tengo', hijos, 'hijos\n'
print '\n\n\tQuiero tener un hijo? (sn)'
respuesta = raw_input()
while (respuesta <> 'n'):
    newpid = os.fork()
    if newpid == 0:
        #
        # Seccion del hijo
        #
        time.sleep(30)
        print '\t\t\t\t', os.getpid(),' - Me aburro. Me voy a jugar a la PLAY'
        exit(0)
    else:
        #
        # seccion del padre
        #
        hijos = hijos + 1
        print '\t\tTuve un hijo!!!! Tiene el PID: ', newpid

    print '\n\tQuiero tener otro hijo? (s|n)'
    respuesta = raw_input()

print '\nBueno, hasta aca llegue. Me voy a dormir. Ya con', hijos ,'hijos es suficiente'
exit(0)
```

Terminación de procesos

- Ante un (**exit**), se retorna el control al sistema operativo
  - El proceso padre puede recibir un código de retorno (via **wait**)
- Proceso padre puede terminar la ejecución de sus hijos (**kill**)
  - La tarea asignada al hijo se terminó
  - Cuando el padre termina su ejecución
    - Habitualmente no se pemite a los hijos continuar, pero existe la opción.
    - Terminación en cascada

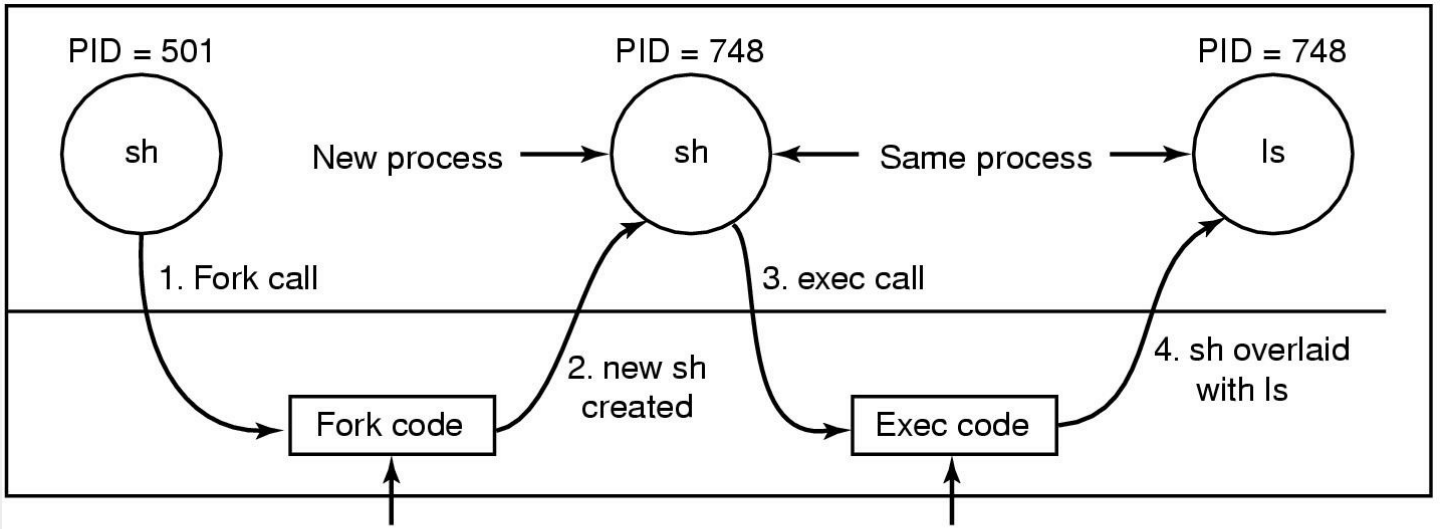
System Call fork, wait y exit

```
#
# El padre espera que terminen sus hijos antes de retirarse
#
import os, time
hijos = 0
print '\n\n\nSoy el PROCESO', os.getpid() , 'y tengo', hijos, 'hijos\n'
while True:
    newpid = os.fork()
    if newpid == 0:
        #
        # Seccion del hijo
        #
        time.sleep(30)
        print '\t(Hijo', os.getpid(), ') - Se va a jugar a la play'
        exit(0)
    else:
        #
        # Seccion del padre
        #
        hijos = hijos + 1
        print '\t(Padre) Tuve un hijo!!!! Se llama', newpid
    if raw_input( ) == 'q': break

print '\n(Padre) - VAYAN A JUGAR A LA PELOTA!!!!'

while hijos > 0:
    os.wait()
    print '\t(Padre) - Joya, uno menos para cuidar!!!\n'
    hijos = hijos - 1

print '\n(Padre) - Listo, se fueron todos, me voy a dormir'
exit(0)
```

Creación de Procesos

- En UNIX:
  - system call **fork()** crea nuevo proceso
  - system call **execve(),** usada después del fork, carga un nuevo programa en el espacio de direcciones.
- En Windows:
  - system call **CreateProcess()** crea un nuevo proceso y carga el programa para ejecución.

Fork / Exec – Ejemplo



Allocate child's process table entry
Fill child's entry from parent
Allocate child's stack and user area
Fill child's user area from parent
Allocate PID for child
Set up child to share parent's text
Copy page tables for data and stack
Set up sharing of open files
Copy parent's registers to child

Find the executable program
Verify the execute permission
Read and verify the header
Copy arguments, environ to kernel
Free the old address space
Allocate new address space
Copy arguments, environ to stack
Reset signals
Initialize registers

Creación y Terminación de Procesos

- Un ejemplo de una CLI (command line interface) o shell

```
1  while (TRUE) {                                  /* repeat forever */
2    type_prompt( );                               /* display prompt */
3    read_command (command, parameters)            /* input from terminal */
4
5    if (fork() != 0) {                            /* fork off child process */
6      /* Parent code */
7      waitpid( -1, &status, 0);                   /* wait for child to exit */
8
9    } else {
10     /* Child code */
11     execve (command, parameters, 0);            /* execute command */
12   }
13 }
14
```

System Call fork, execv

```
#
# SHELL
#
import os, time
print '''
-------------------------------------------------------
----------- Esta es la ISO DIR/LS SHELL --------------
-------------------------------------------------------
# exit (para salir)
'''
cmd = raw_input("iso:\> ")
while cmd <> 'exit':

    if cmd == '':
        cmd = raw_input("iso:\> ")
    else:
        newpid = os.fork()
        if newpid == 0:
            # Seccion del hijo
            lista = cmd.split(' ')
            os.execvp(lista[0], lista)
            print "Imprimir AAAAAAAAAA"
            exit(0)
            print "Imprimir BBBBBBBBBB"
        else:
            # Seccion del padre
            #os.wait()
            cmd = raw_input("iso:\> ")
exit(0)
```

Procesos Cooperativos e Independientes

- *Independiente*: el proceso no afecta ni puede ser afectado por la ejecución de otros procesos. No comparte ningún tipo de dato.
- *Cooperativo*: afecta o es afectado por la ejecución de otros procesos en el sistema.

Para qué sirven los procesos cooperativos?

- ☑ Para compartir información (por ejemplo, un archivo)
- ☑ Para acelerar el cómputo (separar una tarea en sub-tareas que cooperan ejecutándose paralelamente)
- ☑ Para planificar tareas de manera tal que se puedan ejecutar en paralelo.

System Calls - Unix

System call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, opts)	Wait for a child to terminate
s = execve(name, argv, envp)	Replace a process' core image
exit(status)	Terminate process execution and return status
s = sigaction(sig, &act, &oldact)	Define action to take on signals
s = sigreturn(&context)	Return from a signal
s = sigprocmask(how, &set, &old)	Examine or change the signal mask
s = sigpending(set)	Get the set of blocked signals
s = sigsuspend(sigmask)	Replace the signal mask and suspend the process
s = kill(pid, sig)	Send a signal to a process
residual = alarm(seconds)	Set the alarm clock
s = pause( )	Suspend the caller until the next signal

Syscalls de Procesos

System calls - Windows

Win32 API Function	Description
CreateProcess	Create a new process
CreateThread	Create a new thread in an existing process
CreateFiber	Create a new fiber
ExitProcess	Terminate current process and all its threads
ExitThread	Terminate this thread
ExitFiber	Terminate this fiber
SetPriorityClass	Set the priority class for a process
SetThreadPriority	Set the priority for one thread
CreateSemaphore	Create a new semapahore
CreateMutex	Create a new mutex
OpenSemaphore	Open an existing semaphore
OpenMutex	Open an existing mutex
WaitForSingleObject	Block on a single semaphore, mutex, etc.
WaitForMultipleObjects	Block on a set of objects whose handles are given
PulseEvent	Set an event to signaled then to nonsignaled
ReleaseMutex	Release a mutex to allow another thread to acquire it
ReleaseSemaphore	Increase the semaphore count by 1
EnterCriticalSection	Acquire the lock on a critical section
LeaveCriticalSection	Release the lock on a critical section

Syscalls de Procesos