Sistemas Operativos 72.11

Procesos

ITBA
Instituto Tecnológico de Buenos Aires

Procesos

Modelo de procesos

- Todo el software ejecutable se organiza en **procesos secuenciales** o simplemente **procesos**

- **Proceso**: Abstracción de programa en ejecución
- **Programa**: Almacenado en disco, no hace nada.
- ¿Un programa corriendo 2+ veces? -> optimización

- Conceptualmente un proceso tiene su propio CPU
- Posee sus propios registros y variables

- Proveen pseudo concurrencia incluso con un único CPU
- Se lo suele llamar pseudo paralelismo
- Paralelismo real: Múltiples CPUs compartiendo memoria física

- Facilita pensar en una colección de procesos corriendo pseudo paralelamente en lugar de pensar en los switches -> multiprogramming
- Este switch no es uniforme ni reproducible -> supuestos sobre tiempo de ejecución

- Ejemplo switching

Procesos

Modelo de procesos

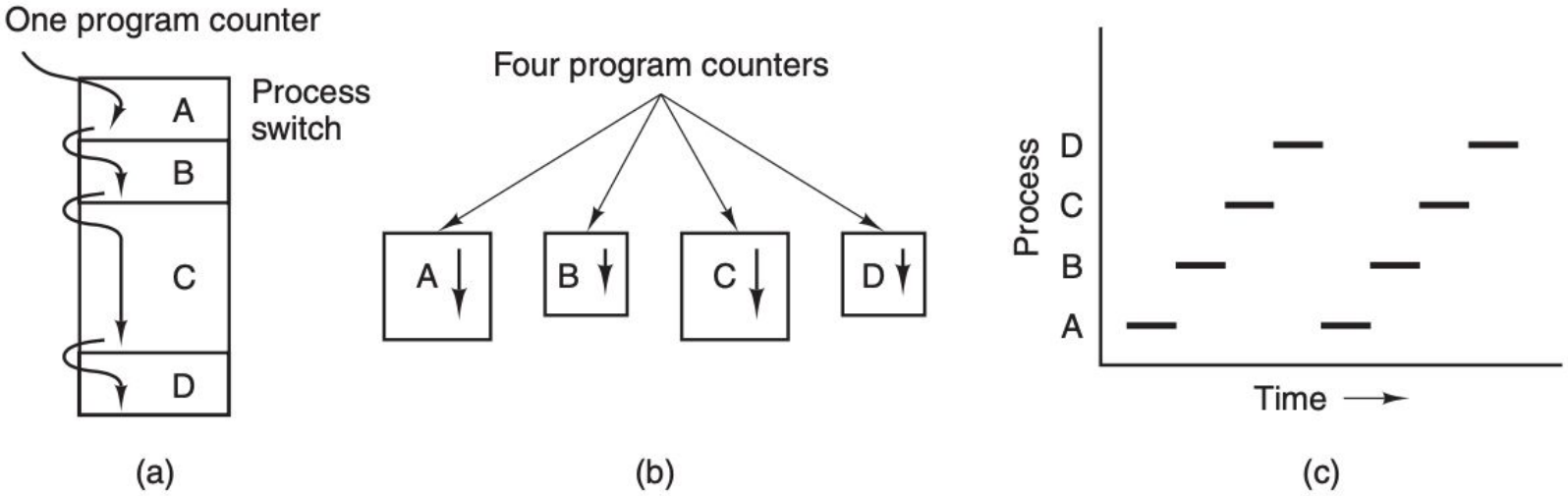


Figure 2-1. (a) Multiprogramming four programs. (b) Conceptual model of four independent, sequential processes. (c) Only one program is active at once.

Procesos

Modelo de procesos

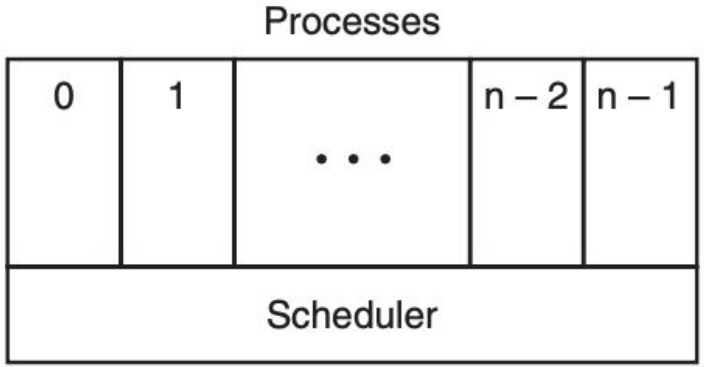
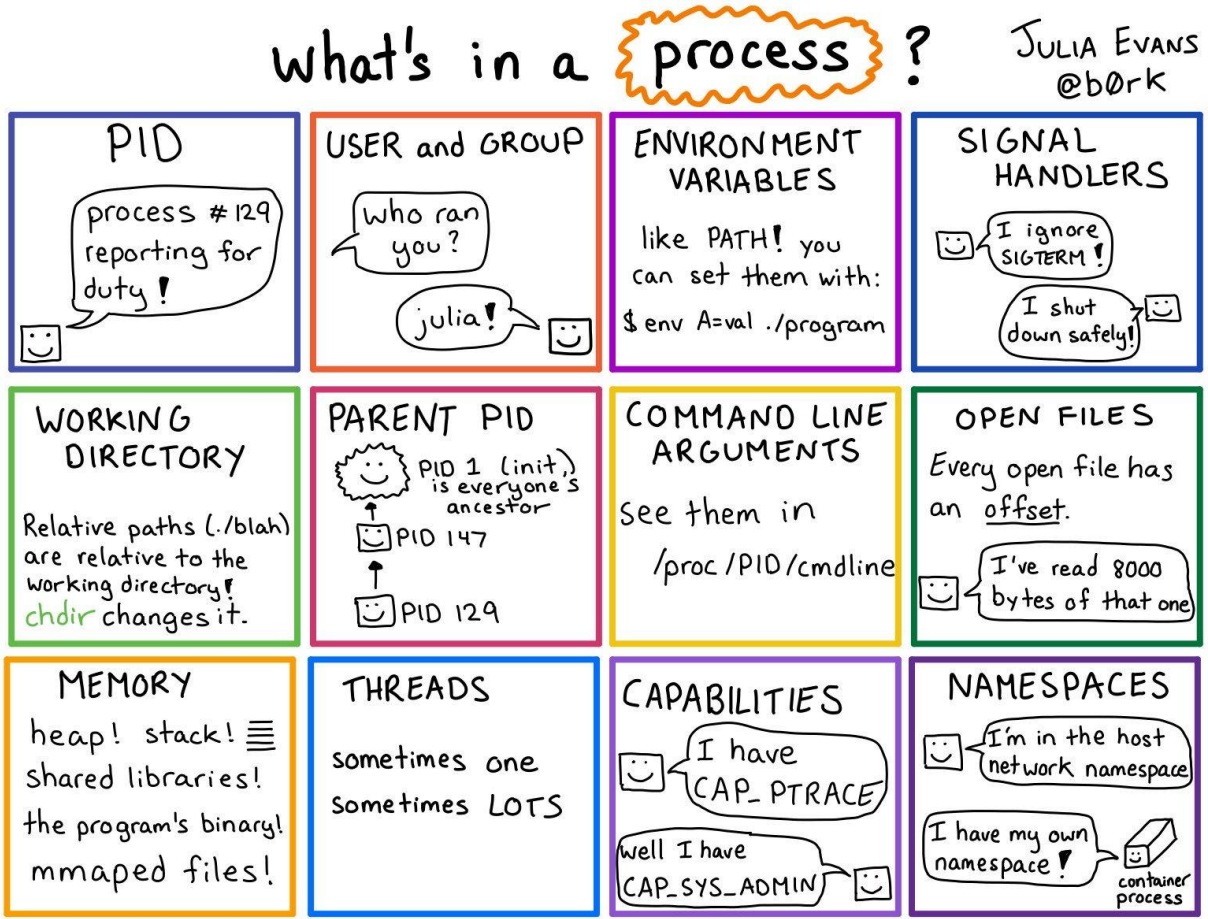


Figure 2-3. The lowest layer of a process-structured operating system handles interrupts and scheduling. Above that layer are sequential processes.

Procesos



Procesos

Creación de procesos

- Sistemas simples pueden no requerir la creación de procesos -> microondas
- Sistemas de propósito general lo requieren -> ¿con quién hay que hablar?

Casos que lo requieren

- Cuando se inicia un sistema operativo
  - Daemons
- Un proceso solicita la creación de otro proceso
  - Formular el problema en términos de problemas relacionados.
  - Script bash y comandos UNIX
- Un usuario solicita crear un nuevo proceso
  - Terminal, Interfaz gráfica (GUI)
- Inicio de un trabajo por lotes
  - Aplicable a mainframes, cuando hay recursos disponibles se toma el siguiente trabajo

Técnicamente en todos estos casos un proceso ejecuta otro proceso mediante una syscall, pero…

Procesos

Creación de procesos - UNIX

fork

- Crea un clon exacto del caller. Luego de fork, los dos procesos (el padre y el hijo) tienen la misma memoria, las mismas variables de ambiente y los mismos archivos abiertos.

execve

- Cambia completamente la imagen del proceso. Esto incluye memoria (heap, stack, código), registros (de propósito general y especiales). Pero preserva archivos abiertos entre otras cosas.

La creación en 2 etapas facilita le permite al hijo manipular los file descriptors antes del exec.

Procesos

Creación de procesos - Win32

createProcess (10 parámetros)

- El programa a ejecutar
- Parámetros
- Argumentos
- Atributos de seguridad y control
- Prioridad
- Ventana a asociar con el proceso

+100 syscalls extras para administrar y sincronizar procesos

- En ambos casos (UNIX y Win32) los procesos creados tienen su propio espacio de direcciones, diferente al del padre

Procesos

Terminación de procesos

Casos en los que ocurre

- Salida normal (voluntaria)
  - exit(0) / return 0;
  - _start()
- Salida por error (voluntaria).
  - exit(!= 0) / return != 0;
  - Colores zsh
- Error fatal (involuntario)
  - Instrucciones inválidas, división por 0, memoria inexistente
  - Señales
- Muerto por otro proceso (involuntario)
  - kill
  - Permisos -> usuarios -> kernel/user mode?

¿Y los procesos creados por este proceso?

Demo

Procesos

Jerarquía de procesos

- Un proceso y su creador preservan una relación
- Generalmente padre / hijo

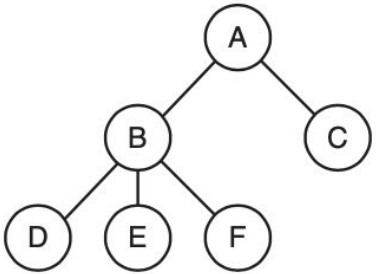
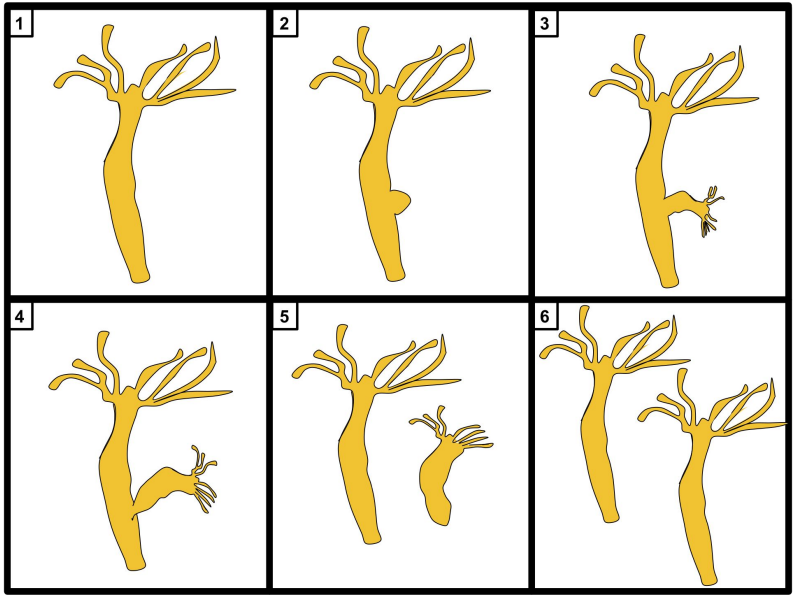


Figure 1-13. A process tree. Process *A* created two child processes, *B* and *C*. Process *B* created three child processes, *D*, *E*, and *F*.

Procesos

Jerarquía de procesos

- Diferencia con animales y plantas que se reproducen de a pares
- Parecido con la hydra -> gemación y hermafroditismo



https://es.wikipedia.org/wiki/Hydra_(animal)

Procesos

grupo de procesos - UNIX

- Un padre y sus descendientes forman un grupo de procesos
- Recepción de señales por grupo, por ejemplo el teclado
- Atención de señales independiente

- Proceso init
- init es la raíz de este árbol -> analogía object

Procesos

Jerarquía de procesos - Win32

- No existe una jerarquía de procesos
- El padre recibe un handle para manejar al proceso
- El handle se puede transferir a otro proceso

Procesos

Estados de procesos

- Running (usando el CPU en ese instante)
- Ready (ejecutable; detenido temporalmente para dejar que se ejecute otro proceso)
- Blocked (no se puede ejecutar hasta que ocurra algún evento externo)

Demo ps

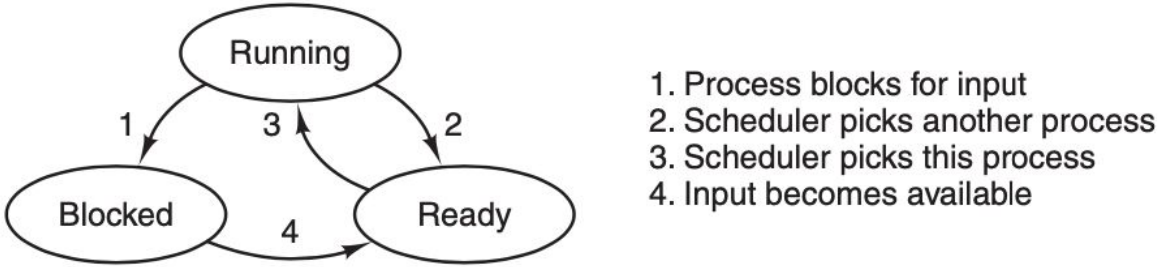


Figure 2-2. A process can be in running, blocked, or ready state. Transitions between these states are as shown.

Procesos

Implementación de procesos

- Se mantiene una tabla de procesos
- Cada entrada se denomina Process Control Block (PCB)

- Información necesaria para pasar de ready -> runnning -> blocked

Procesos

Implementación de procesos

Process management	**Memory management**	**File management**
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Figure 2-4. Some of the fields of a typical process-table entry.

Procesos

an amazing directory: /proc

Julia Evans
@b0rk

Every process on Linux has a PID (process ID) like 42.

In /proc/42, there's a lot of VERY USEFUL information about process 42

/proc/PID/cmdline

command line arguments the process was started with

/proc/PID/exe

symlink to the process's binary. magic: works even if the binary has been deleted!

/proc/PID/environ

all of the process's environment variables

/proc/PID/status

Is the program running or asleep? How much memory is it using? And much more!

/proc/PID/fd

Directory with every file the process has open!

Run `$ ls -l /proc/42/fd` to see the list of files for process 42.

These symlinks are also magic & you can use them to recover deleted files ♥

/proc/PID/stack

The kernel's current stack for the process. Useful if it's stuck in a system call

/proc/PID/maps

List of process's memory maps. Shared libraries, heap, anonymous maps, etc.

and more

Look at

man proc

for more information!

Procesos

Implementación de procesos

Ilusión de múltiples procesos secuenciales under the hood:

1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector.
3. Assembly-language procedure saves registers.
4. Assembly-language procedure sets up new stack.
5. C interrupt service runs (typically reads and buffers input).
6. Scheduler decides which process is to run next.
7. C procedure returns to the assembly code.
8. Assembly-language procedure starts up new current process.

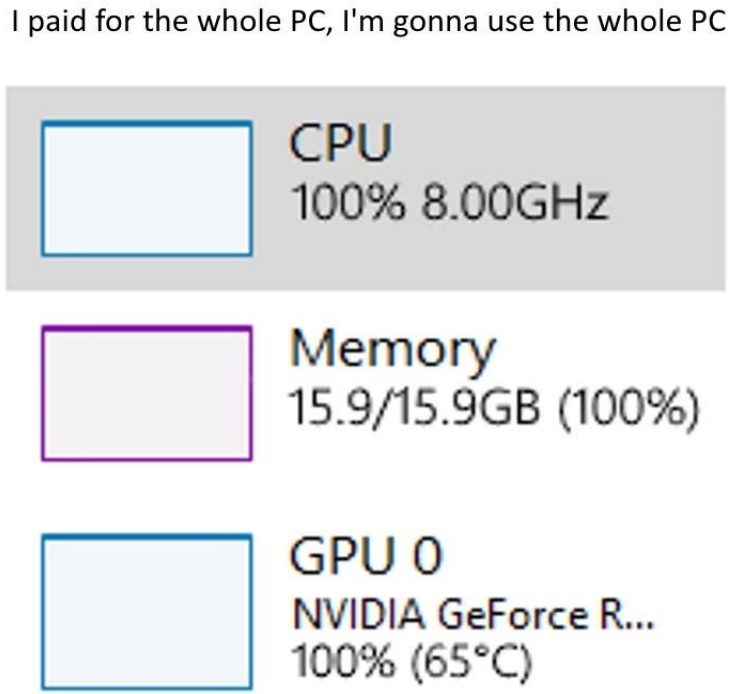
Figure 2-5. Skeleton of what the lowest level of the operating system does when an interrupt occurs.

Ocurre múltiples veces, pero es transparente para el proceso

Procesos

Modelando multiprogramación

- Permite mejorar la utilización del CPU



Procesos

Modelando multiprogramación

Análisis (muy) simple

- Si 1 proceso usa el CPU el 20% del tiempo
- Entonces 5 procesos mantendrían el CPU al 100%

¿Realista?

Procesos

Modelando multiprogramación

Análisis (no tan) simple

- Probabilísticamente
- Un proceso pasa una proporción p del tiempo esperando por I/O (bloqueado)
- Con n procesos en memoria al mismo tiempo, la probabilidad de que los n estén esperando (CPU libre) es p^n.
- La utilización del CPU se puede expresar como

$$\text{Utilización del CPU} = 1 - p^n$$

Procesos

Modelando multiprogramación

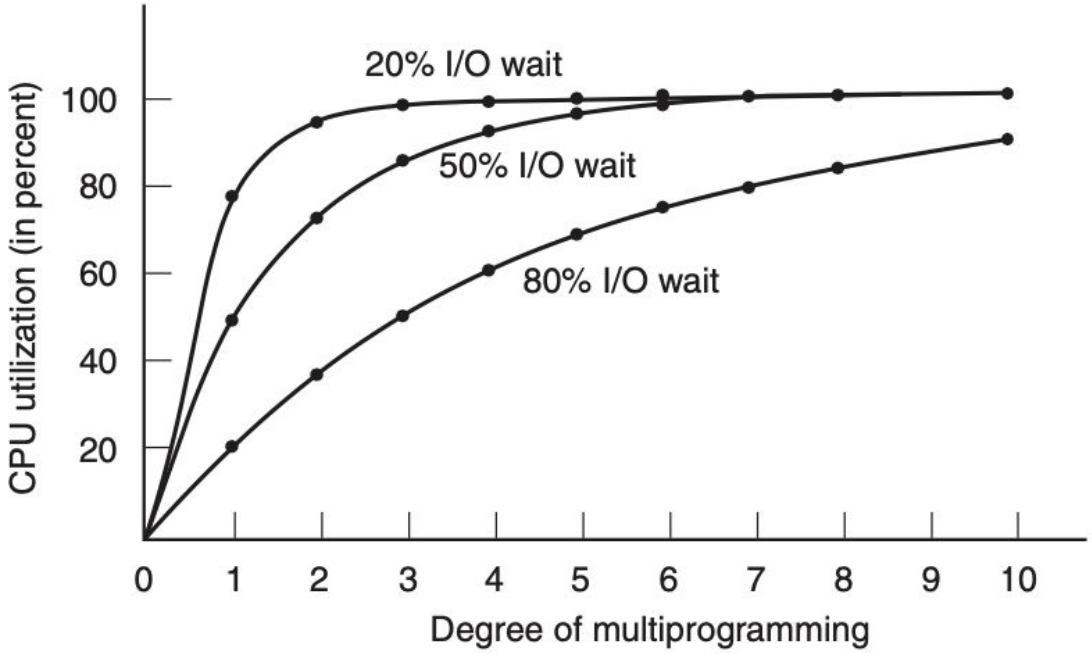


Figure 2-6. CPU utilization as a function of the number of processes in memory.

- ¿Qué tan raro es estar el 80% del tiempo esperando por I/O?
- ¿Realista?
- Vamos al punto. La multiprogramación “garpa”

Ejercicio 1

El comando **`type`** nos indica el tipo de comando que se pasa como argumento. Los posibles resultados son

- alias
- keyword
- function
- builtin
- file

1. Provea un ejemplo de cada uno. Por ejemplo, **`type`** es de tipo builtin, es decir

   ```
   $ type type
   type is a shell builtin
   ```

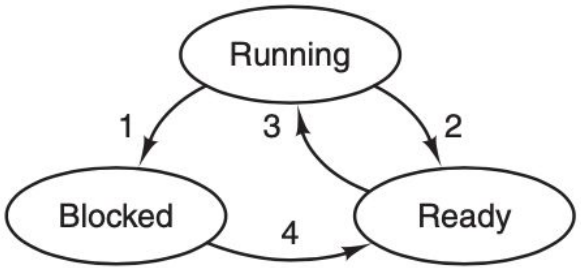
   Para más información sobre type consulte (man no nos puede ayudar en este caso)

   ```
   $ help type
   ```

2. ¿Por qué el comando **`cd`** es builtin?
3. Si no fuera builtin, ¿qué modificaciones considera necesarias para proveer la misma funcionalidad.

Ejercicio 2

En el diagrama de estados de un proceso se presentan 3 estados, por tanto podrían existir 8 transiciones. Analice por qué no se presentan las transiciones faltantes.



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Figure 2-2. A process can be in running, blocked, or ready state. Transitions between these states are as shown.

Ejercicio 3

El comando **`time`** reporta el tiempo utilizado por un proceso discriminado por *real* (tiempo total), *user* (tiempo ejecutando en user space) y *sys* (tiempo ejecutando en kernel space).

1. Ejecute **`time bash`** y luego ejecute comandos como si estuviera usando la shell, por ejemplo, listar el directorio (ls), compilar un programa (gcc), ejecutarlo, copiar un archivo (cp), listar procesos (ps), etc. Al finalizar, presione *CTRL + D* para enviar el EOF y que finalice el intérprete de comandos (bash).
    a. Analice las proporciones de *user* y *sys* respecto de *real*.
    b. ¿Qué cree que hace el proceso bash el resto del tiempo?
2. Ejecute **`time cat`** y tipee un texto lo más rápido que pueda. Al finalizar presione *CTRL + D* para enviar el EOF y que finalice el comando **`cat.`**
    a. Analice las proporciones de *user* y *sys* respecto de *real*.
3. Cree un archivo muy grande, por ejemplo, ejecutando **`cat /dev/random > big`** durante unos 10 segundos e interrumpiendo con *CTRL + C*. (Con **`ls -lh`** puede consultar el tamaño). Luego ejecute **`time md5sum big`**.
    a. Analice las proporciones de *user* y *sys* respecto de *real*.
4. Cree un programa simple cuyo tiempo de ejecución es kernel space predomine
    a. Analice las proporciones de *user* y *sys* respecto de *real*.

Ejercicio 4

Investigue qué información puede encontrar en el directorio /proc. **`man proc`** es un buen comienzo

Ejercicio 5

Una computadora particular puede alojar 5 procesos en su memoria principal. Estos procesos están bloqueados por I/O la mitad del tiempo. ¿Qué fracción del tiempo de CPU es desperdiciada?

Ejercicio 6

Una computadora particular tiene 4GB de RAM, de los cuales 512 MB son ocupados por el sistema operativo. Los procesos (por simplicidad) ocupan 256 MB c/u y tienen las mismas características. Si el objetivo es lograr un 99% de utilización de CPU, ¿Cuál es la máxima proporción de tiempo que puede pasar un proceso esperando por I/O?

Ejercicio 7

Considere un sistema con un grado de multiprogramación de 6 (i.e 6 procesos en memoria al mismo tiempo). Asuma que cada proceso pasa el 40% del tiempo esperando por I/O. ¿Cuál es la utilización de CPU?