PROCESOS

Podemos usar trabajo o proceso indistintamente, pero se prefiere este último

EL PROCESO

Contador de programa

Actividad actual

Contenidos de los registros del procesador

Es un programa en ejecución

Pila del proceso

Código

Cúmulo de memoria

Datos temporales

Sección de datos

Un programa en sí no es un proceso, es una entidad pasiva, archivo que contiene una lista de instrucciones. Un proceso es una entidad activa. El programa se convierte en proceso cuando se carga en memoria un ejecutable.

Puede haber varios procesos asociados al mismo programa, cada proceso se considera una secuencia de ejecución independiente.

ESTADO DEL PROCESO

A medida que se ejecuta un proceso, este va cambiando de estado. Puede estar en los siguientes estados:

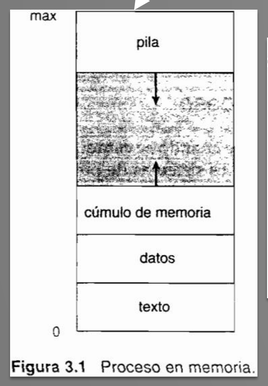


Figura 3.1 Proceso en memoria.

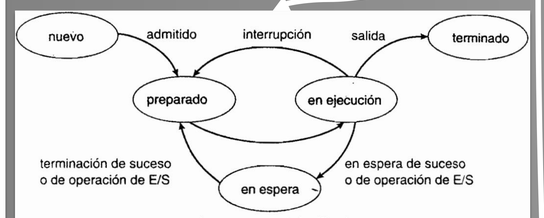


Figura 3.2 Diagrama de estados de un proceso.

- **Nuevo**. El proceso está siendo creado.
- **En ejecución**. Se están ejecutando las instrucciones.
- **En espera**. El proceso está esperando a que se produzca un suceso (como la terminación de una operación de E/S o la recepción de una señal).
- **Preparado**. El proceso está a la espera de que le asignen a un procesador.
- **Terminado**. Ha terminado la ejecución del proceso.

BLOQUE DE CONTROL

Cada proceso se representa en el SO mediante un PCB (process control block). Contiene muchos elementos de información asociados con un proceso específico, como:

- Estado del proceso
- Contador de programa
- Registros de la CPU
- Información del estado de E/S
- Información contable
- Información de gestión de memoria
- Información de planificación de la CPU

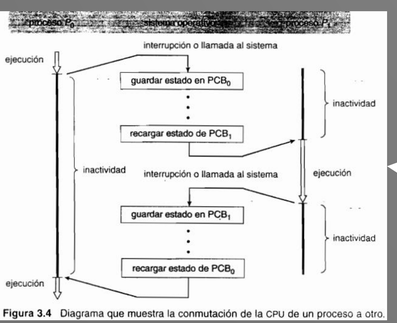


Figura 3.4 Diagrama que muestra la conmutación de la CPU de un proceso a otro.

HEBRAS

El modelo visto hasta ahora implica que solo hay una hebra de ejecución para el proceso. Muchos SO modernos permiten que un proceso tenga múltiples hebras.

PLANIFICACIÓN DE PROCESOS

Para que la multiprogramación y los sistemas de tiempo compartido funcionen, el planificador de procesos selecciona un proceso disponible para ejecutar el programa en la CPU.

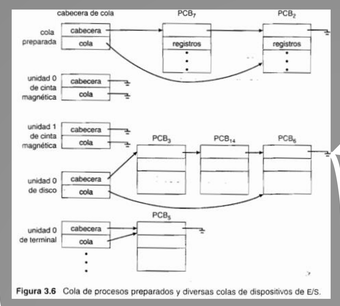


Figura 3.6 Cola de procesos preparados y diversas colas de dispositivos de E/S.

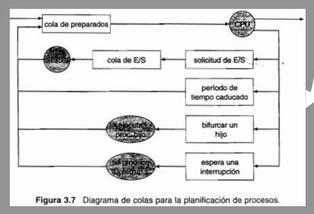


Figura 3.7 Diagrama de colas para la planificación de procesos.

COLAS DE PLANIFICACIÓN

Conforme entran los procesos al sistema, se colocan en una cola de trabajos que contiene todos los procesos del sistema. Aquellos que residen en la memoria principal y están preparados para ejecutarse se mantienen en una lista denominada "cola de procesos preparados". Generalmente dicha cola se almacena en forma de lista enlazada. La cabecera contiene punteros al primer y último bloques de PCB de la lista. Cada PCB incluye un campo de puntero al siguiente.

Cada proceso nuevo se coloca inicialmente en la cola de procesos preparados, donde espera a ser despachado. Una vez asignado a la CPU, puede pasar lo siguiente:

El proceso podría ser desalojado de la CPU por alguna interrupción y puesto de nuevo en la cola

El proceso podría ejecutar una solicitud de E/S y ser colocado en una cola de E/S

El proceso podría crear un nuevo subproceso y esperar a que este termine

PLANIFICADORES

El proceso de selección se realiza mediante un planificador apropiado

El planificador a largo plazo o planificador de trabajos:

Selecciona procesos de esta cola y los carga en memoria para su ejecución

El planificador a corto plazo o planificador de la CPU:
Selecciona entre los procesos que ya están preparados para ser ejecutados y asigna la CPU a uno de ellos

La principal diferencia entre estos dos planificadores se encuentra en la frecuencia de ejecución

El planificador a largo plazo se ejecuta mucho menos frecuente.

El planificador a corto plazo debe seleccionar un nuevo proceso para la CPU frecuentemente.

OPERACIONES SOBRE LOS PROCESOS

```
#include <stdio.h>
#include <windows.h>

int main (VOID)
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

   // asignar memoria
   ZeroMemory(&si, sizeof(si));
   si.cb = sizeof(si);
   ZeroMemory(&pi, sizeof(pi));

   // crear proceso hijo
   if (!CreateProcess(NULL,   //  utilizar línea de comandos
     "C:\ WINDOWS\\system32\\mspaint.exe",  //  línea de comandos
     NULL,  //  no hereda descriptor del proceso
     NULL,  // no hereda descriptor de la hebra
     FALSE,  //  inhabilitar herencia del descriptor
     0,      //  no crear indicadores
     NULL,  //  usar bloque de entorno del padre
     NULL,  //  usar directorio existente del padre
     &si,
     &pi))
   {
     fprintf(stderr,  "Fallo en la creación del proceso");
     return -1;
   }
   //  el padre espera hasta que el hijo termina
   WaitForSingleObject(pi.hProcess, INFINITE);
   printf "Hijo completado");

   //  cerrar descriptores
   CloseHandle(pi.hProcess);
   CloseHandle(pi.hThread);
}
```

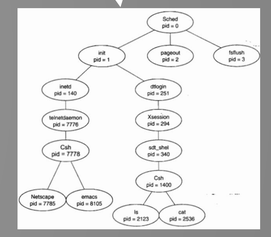
Figura 3.12 Creación de un proceso separado usando la API de Win32.

Los procesos pueden ejcutarse de forma concurrente y pueden crearse y eliminarse dinámicamente.

Por esto, los sistemas deben proporcionar un mecanismo para la creación y la terminación de procesos. dinámicam ente.

CREACIÓN DE PROCESOS

Un proceso puede crear otros varios procesos. A este proceso se le denomina PROCESO PADRE y los procesos que crea se consideran PROCESOS HIJOS y cada uno de ellos puede crear a varios procesos, lo que da pie a lo que se llama **"Arbol de Procesos"**



TERMINACIÓN DE PROCESOS

Un proceso termina cuando ejecuta su última instrucción y pide al sistema operativo que lo elimine usando la llamada al sistema exit(). El sistema operativo libera la asignación de todos los recursos del proceso, incluyendo las memorias física y virtual, los archivos abiertos y los búferes de E/S.

Un proceso puede causar la terminación de otro proceso a través de la adecuada llamada al sistema, normalmente, dicha llamada sólo puede ser invocada por el padre del proceso que se va a terminar.

Un padre puede terminar la ejecución de uno de sus hijos por diversas razones, como por ejemplo:

-> El proceso hijo ha excedido el uso de algunos de los recursos que se le han asignado. Para determinar si tal cosa ha ocurrido, el padre debe disponer de un mecanismo para inspeccionar el estado de sus hijos.

-> La tarea asignada al proceso hijo ya no es necesaria

->El padre abandona el sistema, y el sistema operativo no permite que un proceso hijo continúe si su padre ya ha terminado

Cambio de Contexto

Cuando se produce una interrupción en el sistema tiene que guardar el contexto actual del proceso que se esta ejecutando en la CPU de modo que se puede restaurar cuando su procesamiento concluya.

La conmutación de la CPU a otro proceso requiere una salvaguarda del estado del proceso actual y una restauración del estado a otro proceso diferente.

Cuando se produce un cambio de texto, el kernel guarda el contexto del proceso antiguo en la PCB y carga el contexto almacenado del nuevo proceso que se ha decidido ejecutar

El tiempo dedicado al cambio de contexto es desperdiciado, dado que el sistema no realiza ningún trabajo útil durante la conmutación

El tiempo empleado en los cambios de contextos depende fundamentalmente del soporte de hardware.

EQUIPO 6
Mendoza García Eliú Eduardo
Colín Ramiro Joel
Maldonado Cerón Carlos
Hernández Reyes Julio Cesar